(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
25 May 2001 (25.05.2001)

PCT

(10) International Publication Number
WO 01/37170 A2

(51) International Patent Classification[7]: G06F 17/60

(21) International Application Number: PCT/US00/31221

(22) International Filing Date:
15 November 2000 (15.11.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/443,378 19 November 1999 (19.11.1999) US

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier application:
US 09/443,378 (CON)
Filed on 19 November 1999 (19.11.1999)

(71) Applicant *(for all designated States except US)*: **SCIENCE MANAGEMENT CORPORATION** [US/US]; Suite #103, 1803 Research Blvd., Rockville, MD 20850 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **HAUG, Ralf** [DE/US]; 7122 D Rockridge Lane, Alexandria, VA 22315 (US). **BHUTT, Sanjeev** [ZA/US]; 1567 Swallow Drive, Brentwood, MO 63144 (US). **GRIMLEY, Benjamin, Lewis** [DE/US]; 610 Poplarwood Place, Gaithersburg, MD 20877 (US). **LEVER, Farrel, Joseph** [ZA/US]; Apartment 1813, 10101 Grosvenor Place, Rockville, MD 20852 (US). **ORBACH, Tamir** [ZA/US]; 10113 Vanderbilt Circle, Rockville, MD 20850 (US). **POLLARD, Michael, Edward** [US/US]; Suite 829, 4701 Willard Avenue, Chevy Chase, MD 20815 (US).

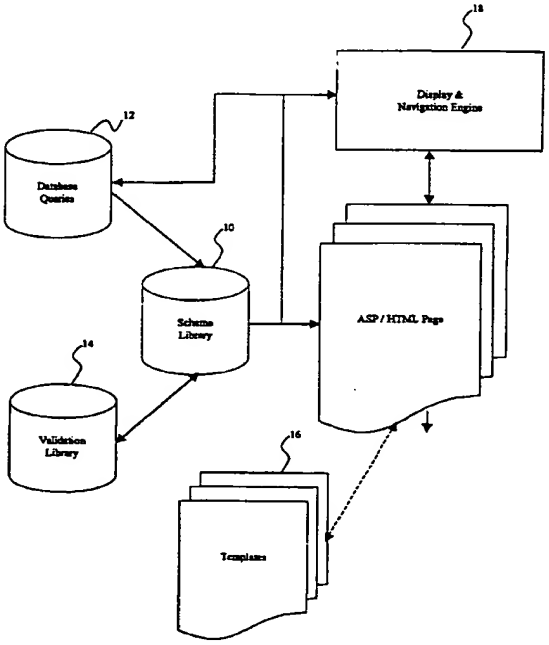
(74) Agent: **ELLIS, William, T.**; Foley & Lardner, Suite 500, 3000 K Street, N.W., Washington, DC 20007-5109 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE,

[Continued on next page]

(54) Title: FORMS CREATION METHOD AND E-COMMERCE METHOD



(57) Abstract: An integrated development method for creating forms, comprising the steps of: storing at least one schema for a form in a first file at a computing resource, wherein the schema comprises minimal user interface representations; accessing at least one template of a form in at least one second file separate from the first file at a computing resource, wherein each of the templates comprises a user interface representation of the form; selecting one of the at least one template and one of the at least one schema; and creating at least one linking file that links the selected schema and the selected template. A system and method for performing actions in an e-commerce environment is also provided.

WO 01/37170 A2

IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:

— *Without international search report and to be republished upon receipt of that report.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

FORMS CREATION METHOD AND E-COMMERCE METHOD

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates generally to forms creation software and e-commerce, and more particularly, to a method, system and software for creating and maintaining highly scalable and fully functional, multi-step web-based forms and the creation of a multi-tiered e-commerce method and system.

2. Description of the Related Art

There is a specific problem with current forms development software in that they are not scalable, and are not easily implemented across different computer systems. Such prior art systems typically require experienced programmers to design the forms on those systems.

There is also the problem of controlling the workflow in e-commerce operations, particularly those e-commerce operations using multiple components.

These problems have not been solved in the prior art.

SUMMARY OF THE INVENTION

Briefly, in a first aspect the present invention comprises an integrated development method for creating forms, including the steps of: storing at least one schema for a form in a first file at a computing resource, wherein the schema comprises minimal user interface representations; accessing at least one template of a form in at least one second file separate from the first file at a computing resource, wherein each of the templates comprises a user interface representation of the form; selecting one of the at least one template and one of the at least one schema; and creating at least one linking file that links the selected schema and the selected template.

In a further aspect of the invention, a method is provided to publish forms comprising the steps of: creating a schema; providing a list of predefined templates; selecting a template; selecting a publishing destination; and linking the schema to the selected template.

In yet a further aspect of the present invention, a method is provided for creating a regular expression file comprising the steps of: inputting and presenting a name for a regular expression; inputting and presenting a pattern for a regular expression that is to be used in a validation engine; inputting and presenting an error message for user inputs that do not match the pattern; inputting at least one test case; and comparing the test case to the pattern; and presenting a message in response to the comparison step.

In a further aspect of the present invention, a navigation engine program product is provided, comprising: a computer usable medium having computer readable program code embodied therein comprising: first code for transforming data-centric hierarchical storage and retrieval mechanism data to presentation language; second code for receiving data in presentation language and converting the received data to the data-centric hierarchical storage and retrieval mechanism; third code for performing attribute based validation on the data; fourth code for providing navigation among present pages; and fifth code for saving the data in response to a command.

In yet a further aspect of the present invention, a method for processing forms on a user-interface program is provided, comprising the steps of: receiving an engine and a file with a form including a schema and a template in a data-centric hierarchical storage and retrieval mechanism; transforming with the engine the form from the data-centric hierarchical storage and retrieval mechanism into presentation language and presenting the form at the user-interface program; receiving user input data in the presentation language; converting the user input data to the data-centric hierarchical storage and retrieval mechanism; performing attribute based validation on the converted user input data; incorporating any error detected in the validation step in the data-centric hierarchical storage and retrieval mechanism into the form schema or separate form schema; converting the any error into presentation language and presenting an error message associated therewith to the user; allowing the user to correct the user input data, if necessary; and saving the user input data.

In a yet further aspect of the present invention, a runtime environment method for an electronic forms applications is provided, comprising the steps of: receiving a forms request from a user-interface program; determining if the user-interface program is compatible or not compatible with a data-centric hierarchical storage and retrieval mechanism; sending from a computing resource to the user-interface program, if the user-interface program is compatible with the language, a schema file containing form

information to be inserted into a form template and a form navigation engine and a form input validation engine; if the user-interface program is not compatible with the storage and retrieval mechanism, sending user input information to the computing resource for processing and sending a presentation language version of form pages resulting from the computing resource processing back to the user-interface program.

In a yet further aspect of the present invention, a method for creating a form file is provided comprising the steps of: storing in a form file a path to a schema in a first file at a storage location; storing in the form file a path to a template in a second file different from the first file at a storage location; and storing the form file at a user location.

In a yet further aspect of the present invention, a method for performing customized actions in a computing environment is provided, comprising the steps of: providing at least one selector file, each selector file containing at least one selector pattern and associated with an action set that contains at least one action, wherein the at least one action contains at least an eAgent ID and an associated action document ID; obtaining a document for processing from a queue; searching the document for a match to at least one of the selector patterns of the selector files; if a match is found for one of the selector patterns, executing the at least one action in the Actionset associated with the matched selector file, including loading an eAgent specified by the eAgent ID; and the eAgent reading an action document specified by said associated action document ID and performing an action specified therein.

In yet a further aspect of the present invention, a system for generating forms in a computing environment is provided, comprising: a schema database; a template database separate from the schema database; at least one linkage file, each containing code to link a selected schema with a selected template; a database of validation patterns separate from the schema database and the template database; and a database of query information separate from the schema database and the template database.

In yet a further aspect of the present invention, a system for performing actions on documents in a computing environment is provided, comprising: a selector file database containing a plurality of selector files; an Actionset database containing a plurality of Actionsets, wherein each selector file is associated with exactly one Actionset; an eAgent specific action document database containing a plurality of action documents that are

containing code to link a selected schema from the schema database with a selected template from the template database; fourth code for facilitating a database of validation patterns separate from the schema database and the template database; and fifth code for facilitating a database of query information separate from the schema database and the template database.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a schematic block diagram showing the system architecture for the forms creation system and method of the present invention.

Fig. 2 is a static block diagram of the overall invention architecture, including the forms creation system and method in combination with the e-commerce system.

Fig. 3 is a process workflow block diagram for a client-side processing software design.

Fig. 4 is a workflow block diagram for a server-side processing software design.

Fig. 5 is a process workflow block diagram of a forms publishing software process.

Fig. 6 is a static block diagram of an eAgent architecture.

Fig. 7 is a process workflow block diagram for the eAgent architecture software.

Fig. 8 is a static block diagram of an integrated development system for an e-commerce architecture.

Fig. 9 is a schematic diagram of a forms schema.

Fig. 10 is a schematic diagram of a regular expressions schema.

Fig. 11 is a schematic diagram of a query schema.

Fig. 12 is a schematic diagram of a templates schema.

Fig. 13 is a schematic diagram of a template schema.

Fig. 14 is a schematic diagram of a publishing folder schema.

Fig. 15 is a schematic diagram of an integrated development environment schema.

Fig. 16 is a schematic diagram of an XML selector schema.

Fig. 17 is a schematic diagram of an Actionset schema.

Fig. 18 is a schematic diagram of an action query eAgent schema.

Fig. 19 is a schematic diagram of a queue config schema.

Fig. 20 is a schematic block diagram of a system for performing actions on documents.

- An ActiveX based integrated development environment (IDE), for manipulation of XML files over the Internet using HTTP and the ability to associate the present invention with a template over the web - all using a graphical user interface (GUI). This obviates the need for the forms designer to learn any of the above technologies to create and maintain the present invention.

It should be noted that although the preferred embodiment is described in the context of current language types such as XML, HTML, CSS, and the other listed languages and conventions for ease of reader understanding, it is not intended to limit the invention to these languages and conventions. For example, the references to XML can be replaced by any data-centric hierarchical storage and retrieval mechanism; and HTML can be replaced by any appropriate presentation language.

A number of terms that are used throughout the specification are defined below.

Data-centric Hierarchical storage and retrieval mechanism

A data-centric hierarchical storage and retrieval mechanism comprises the ability to store and retrieve hierarchically organized data. It includes both the format in which the data gets stored, as well as how to access the hierarchically organized data.

Action Document

A document that defines parameters for an action in the data-centric hierarchical storage and retrieval mechanism.

Routing Document

A routing document contains a document and tracks actions executed on the document by the eAgents.

Action

An action can contain one or more processes that may or may not be executed by an eAgent.

eAgent

An eAgent is a software component that performs the action on the routing document and on the associated action document.

Queue

A queue can be a folder, mailbox, queue server or any computing resource capable of queuing data.

Code for facilitating a database

As noted above, Metadata or schema are data that describe the aspects of a form that change from form to form, *excluding the exact (graphical) presentation.* Examples include but are not limited to

- Name and caption of a field
- Help text
- What control type to use
- Whether the field is required and any other validation rule
- Whether to provide the user of the form with a lookup list, and the entries of the list.

The following are examples of what would **not** constitute Meta data or be part of the schema.

- Color (background, font, etc.)
- Fonts (type, size, bold, italic, ...)
- Positioning
- Graphical elements, including lines, circles, logos, etc.
- HTML tags for display of input controls, e.g. <INPUT>, etc.

Using this approach allows the separation of content from display, and the possibility to change each one completely independently of the other.

The schema of a form is represented are typically represented in a specific XML or other data-centric hierarchical vocabulary, that includes those elements and attributes needed to describe a form, but not its display implementation.

An extremely simplified example would look like this:

```
<form name="Registration" >
    <field name="FirstName" required="true"/>
    <field name="LastName" required="true"/>
    <field name="eMail" required="true"/>
</form>
```

The schema would be created by a forms designer and stored in a schema library 10. The schema library 10 provides the forms designer with the ability to reuse individual sections or fields or the entire schema in other forms. Any storage configuration may be utilized including remote and/or distributed storage accessible over a network, over the Internet, or in any other convenient fashion.

Query Library

In one aspect of the invention, in order to present the forms designer with a list of valid entries to populate form fields, the present invention provides the ability to gather the data for list/lookup from any ODBC/OLE DB compliant database, including SQL Server, Access, Oracle, DB 2, DBase, Informix, Sybase, Microsoft Exchange and others.

Instead of including the necessary information for executing a query directly into the native schema of a form, the present invention uses a query library 12 to store queries and only references them inside the native schema of the forms by the given name of the query. This results in the ability to have a central repository 12 of all database-related queries for ease in making changes and flexibility. Queries can be changed independently of the forms that are using them, and any saved changes take immediate effect.

The queries themselves are defined using a specific XML vocabulary, and the connection information to the databases, in a preferred embodiment, is kept in Microsoft Data Link Files (udl files).

Note that to facilitate the use of the query library to facilitate a central change to a query that is used in a plurality of different schema, the present invention, in a preferred embodiment, stores the schema in the schema library 10 in both a native schema format and a compiled format. In the native schema format, the schema includes one or more links to the query library at those sections of the form fields and sections that are to incorporate a particular query therein. The compiled schema format is also stored and incorporates the query itself, rather than a link to the query.

Updating a given schema with changes made to one or more queries would typically be accomplished simply by recompiling the schema. Alternatively, the updating can also be programmed to be accomplished at runtime.

Validation Library

A validation library of regular expressions 14 is provided that contains a list of regular expressions. 'Regular expressions' is a pattern matching technology, that allows for the comparison of an input string against a pattern (regular expression), and that evaluates whether the pattern is found in the input string. In this way, a variety of validations can be executed when a user inputs data into an electronic form, including checking for valid email, zip, state, phone, SSN, etc. Each regular expression is given a user-friendly name and is called from the schema library 14 by the forms designer using that name. The regular expressions library 14 may conveniently store the regular

expressions in XML or other convenient data-centric hierarchical storage and retrieval mechanism.

The same approach may be used as was used for the query library. The native schema stored at in the schema library 10 may include one or more links to regular expressions stored in the regular expressions library 14. The compiled schema stored in the schema library 10 would include the particular regular expression without the link to the regular expression library 14. Accordingly, regular expressions can be updated by one central change to the regular expression library 14, which change can be propagated through all of the schema in the schema library 10 using that regular expression simply by recompiling the particular schema.

Templates

A Template is a collection of files, that may include ASP, CSS, images and other convenient items, which define the layout of the page in which the forms using that template are rendered. The layout typically includes the company logo and a navigation bar to allow the user to navigate to other pages of the site, plus the look and feel of the page, including colors, fonts, positioning, and other aspects.

The present invention enables leveraging the full power of ASP/HTML/CSS to format the page most ways that the forms designer may wish to display the form. The template files can be designed with any tool that can create ASP/HTML/CSS, including Front Page, Visual InterDev, Dream Weaver or even a simple text editor. The templates that are created are stored in a template library 16.

To make an ASP page a template, a designer simply needs to enter appropriate lines of code into his/her ASP page (done automatically when the form is published to the Web) and create additional descriptions in associated XML files.

The present invention allows the forms designer to easily select a template from the template database 16 with which to associate a schema.

Display and Navigation Engine

Tasks

Also included in the system is a display and navigation engine that performs the following tasks

- Transforms the XML forms into HTML forms. (→ Forms translator)
- Takes the input into HTML forms and transfers the data back into XML forms. (→ Forms extractor)

- Performs validation logic that only allows submission of valid data and notifies users of errors and what needs to be corrected.
- Displays a preview page for user verification of entries.
- Page navigation to allow moving forward and backward, including ensuring that any entered data are saved (i.e., NEXT, PREVIOUS, SUBMIT).
- Saves the submitted form data and relevant schema into a queue (the preferred implementation uses file system as a queue).

In order to accomplish the foregoing tasks several components are used. What they are and how they are used is described in the following sections.

Client-Side vs. Server-Side processing

In a preferred embodiment, there are two versions of the display and navigation engine 18. For clients that use Internet Explorer 5 (IE 5) or other browsers that are compatible with a data-centric hierarchical storage and retrieval mechanism, a substantial portion of the processing may take place on the client-side (browser). For other clients that are not compatible with a data-centric hierarchical storage and retrieval mechanism, all the processing occurs on the server. Both versions share some components / component logic, while some parts are specific to each version.

This preferred embodiment using two types of display and navigation engines is utilized in order to support a substantial number of browsers. Note that the display and navigation engine for IE 5 and other similar type browsers use the client-side functionality to offload processing to the client and thereby obtain improved performance (no server round trips) and higher scalability (less server processing needed).

At the present time Internet Explorer 5 comes with sufficient support for the XML DOM (Data Object Model), XML data islands, XSL and DHTML (Dynamic HTML), which are used to implement a client-side version of the present invention. Future browsers that support these or comparable technologies will also support client-side processing.

Forms Translator

Current browsers have no standard way of displaying any XML or other data-centric hierarchical elements in a meaningful way. The present invention uses a forms translator in the display and navigation engine 18 to transform the XML forms into HTML forms.

The forms translator, in the present design, is implemented primarily as XSL style sheets. The Extensible Stylesheet Language (XSL) allows the transformation of XML into another form of XML, HTML or any other text based format.

The forms translator maps in a standard fashion the relevant XML elements and attributes to HTML elements and attributes, and supports multiple controls, field level help, error messages.

Forms Extractor

Once the data has been entered into the *HTML* form by a user, the data must be extracted again into XML by the display and navigation engine 18. The mechanism used to accomplish this is a naming convention. The format for this convention is as follows:

fld_<controltype>_<sectionname>_<fieldname>

where :

"_" is the delimiter that separates the parts to be parsed

"fld" is the prefix which indicates that a form variable is related to the XML DOM data

<controltype> is a descriptive (not necessarily directly listed in the XML DOM structure, for example "selectmulti" might require a match of controltype attribute='select' and multiplselect attribute='yes') term used to uniquely identify a control type in the XML schema. A control type is a reference to the XML representation of an HTML display type, such as text box or select. This approach allows the forms designer to define custom controls, such as a date picker control, which constitutes a composition of several simple controls.

<sectionname> and <fieldname> are the name identifiers of the section/field.

The server-side version uses the ASP Request object to retrieve the information entered, and using the naming convention above transfers the information into XML. The client-side version uses DHTML, in the preferred embodiment.

The conversion to the data-centric hierarchical storage and retrieval mechanism may occur almost simultaneously with the data entry, or may occur periodically, or may conveniently occur when the user clicks NEXT.

Validation

Most typical validation in the prior art acts directly on the user interface. This approach tightly couples the validation to the display mechanism and offers little flexibility.

The present inventive approach allows the validation to work no matter what user interface has been chosen. This is accomplished via the display and navigation engine 18

by *first* converting the information from the display to the XML or other data-centric hierarchical storage and retrieval mechanism, and only then performing the validation on the XML converted data.

By using this approach, the same source code (or the same logic) can be used by both the client and the server-side version of the validation software, and the present invention is able to support changes in the user interface without having to change the validation logic.

The validation logic in the display and navigation engine 18 itself also does not communicate directly with the user interface once errors have been found. Instead it flags the erroneous field/section by including an error message inside the XML-converted data. This offers the advantage of being able to customize how the error message is displayed to the end user.

The validation component functions to check every field for all validation rules defined and stops processing that field/section once/if the first error has been found.

Another key part about the validation is the ability to use attribute-based validation. Examples of attribute-based validation are "required," "maxlength," "minlength," "minvalue," "maxvalue," and "pattern." All that the forms designer needs to do is specify a value for that attribute, and the validation component will ensure that the validation is performed accordingly, and the display engine will ensure the error is communicated to the end user.

By using this approach, non-programmer forms designers can perform 95% or more of the typical validation on most of the forms without having to do any programming.

Preview Page

The preview page for the display and navigation engine 18 is simply a read only view of the caption and values of the fields entered by the end user. The current implementation of the innovation uses an XSL style sheet for the display.

Page Navigation

Page navigation on the client is handled by local Jscript in a preferred embodiment in order to implement the NEXT, PREVIOUS, AND SUBMIT commands.

Note that the server-side version of the display and navigation engine 18 (for nonXML compatible browsers) needs to keep track of the current section being displayed

to the user. This is accomplished by passing a section number as a hidden textbox for every display.

Maintaining state

The data entered by the end user is transferred into the XML document that contains the schema. For the client-side version of the present invention, there is no other problem associated with maintaining state.

For the server-side version of the invention, additional code has to be added, because HTTP is a connection-less protocol. The solution used by the present invention is to store the XML converted data as a file with a unique identifier on the server, and to pass this unique identifier in a hidden textbox from page to page. In this way any previously entered data is available when navigating among pages.

Submission

The typical action in prior art designs when an end user is submitting form data (which generally will occur when the end user clicks SUBMIT) is to directly and synchronously act on the data, e.g., write to the database, send emails, etc. This approach results in the end user being required to wait until all these actions are completed, and in case of the database being offline, the user losing all of his/her data.

The present invention instead uses a queue-based approach, where all data entered and relevant schema are simply written to a queue, and an acknowledgement is sent to the user for his/her submission.

The queue mechanism has several advantages:

- **Quicker site response time.** The user does not need to wait until the actions based on his input are performed.
- **Queues don't lose data even if the connection between the Business Tier Server and the Database Server is down.** Messages will continue to be queued in the queue, and processing will continue as soon as the connection is restored.
- **Forms a log file of transactions.** Allows for every insert request to be logged.
- **Independent changes**— the processing performed on the submitted data can be changed completely independently from the forms themselves, as long as the structure of the form message remains the same.

It should be noted that in the preferred embodiment, the client-side version (i.e., compatible with XML or other data-centric hierarchical storage and retrieval mechanism) submits its data using the XMLHTTPRequest class from Microsoft's XML parser to send the XML to the web server. This again reduces the processing the server needs to perform, because the transferring of the end user-entered data in the HTML form to XML has already been performed by the client browser.

Use Of XML in The present invention v1.0 Programmatic Creation / Reading of XML files

The present invention provides GUI tools to edit XML files. The XML files themselves are never directly exposed to the forms designer. Using this approach ensures the consistent, error-free creation of the relevant XML files and facilitates use by the forms designer.

All of the tools that are needed to create the XML files are embedded into an IDE for easy access and consistent use.

Server- and Client- side Flowcharts

The following figures are intended to clarify the processes that are occurring within the server-side and client-side implementations of the present invention, while highlighting the differences. The differences are primarily due to how the memory and instruction control and execution is partitioned differently between the different implementations. Both have functionality required of the server and client but differ in where the primary processing occurs.

Flow Descriptions - Process Workflow of client-side processing

Referring now to Fig. 3, there is shown a schematic flow diagram of the present invention for the client-side implementation (XML or data-centric hierarchical storage and retrieval mechanism compatible browsers).

- Server receives ASP or other standard page request.

  It should be noted that instead of ASP, which is the preferred embodiment, any server-side scripting/programming solution, such as JSP, could be used.
- Step A:

  The server-side software processes the request at runtime. This encompasses:
  - Loading the schema.
  - Optional initializing the schema (on the fly modification) by a custom ASP script, i.e. adding a field at runtime.

- Preprocessing the schema for any database lookups.
- Optional initializing values by a custom ASP script, i.e. populating a form with values at runtime.
- Putting out the resulting schema as an XML island as part of the HTML page that is sent to the client
- Sending the full HTML page with the XML island down to the browser.

- Step B:

  After the HTML page and all associated files (XML, XSL, CSS, JavaScript, images) have been loaded, set the first page to be the active page and call step C to display it.

- Step C:

  The client-side runtime software calls XSL style sheets to the schema file to display the active page of the form. It is here where the schema of the form is converted to HTML. The conversion includes form/page/section headers and help text, as well as information regarding input fields. This includes captions, different HTML controls to capture input for them, required icons, error messages in case there have been any errors, etc.

  The transformation/conversion uses dynamic HTML to change the content of the page after it has been loaded or after an end user has navigated to a new page by clicking NEXT or preview. There is no need to roundtrip to the server.

- Step D:

  Here the user enters his/her data and clicks on navigation controls to move forward/backward. The client-side display and navigation software awaits the activation of a navigation control.

- Step E:

  Having activated a navigation control, the client-side runtime software transfers the entered data of the page into the schema, which resides as an XML island inside of the HTML page.

- Step F:

  In this step, validation is executed for all sections/fields of this transferred page. The outcome of the validation is documented inside the schema.

- Steps F1 and F2--Page calculation: (Errors?/Last Page?)
- If an error has been found, then step F1 causes the same page to be displayed again (step C).

- Otherwise in step F2 it is checked whether it is the last page. If it is the last page, then a preview page is shown. (step G)

  Note that the preview is optional. If the preview is disabled for this form instance then the data gets submitted after the last page has been filled and the submit button activated.

- If there is neither an error nor the last page, the active page is being set as requested, i.e. the next or previous one and then displayed (step C)
- Step G:

  Using a style sheet for preview, the information entered in the schema is transformed to an HTML preview page and displayed.
- Step H:

  After having clicked NEXT/SUBMIT on the preview page, the XML schema is transformed into another XML document using an XSL style sheet for this purpose. This allows, for example, reducing the size by only including needed information, digitally signing the document or transforming into any other desired format. The transformed XML is sent to the server. This procedure of scaling down the size of the document (deleting items such as display-related attributes, such as the control type to use, the width of the control, and validation-related attributes, such as the isvalid flag and the errormessage attribute) is used to conserve bandwidth.
- Step I:

  The XML is received on the server and submitted to a message queue (e.g., the file system or MSMQ). Optional post-processing is executed. Then, an acknowledgement is sent back to the browser. Optionally, a redirection to the next form, home page or other page is done.

Flow Descriptions - Process Workflow of server-side processing

Referring now to Fig. 4, there is shown a schematic diagram of the execution flow for server-side processing that will be implemented if the browser is not compatible with a data-centric hierarchical storage and retrieval mechanism such as XML.

- In the initial step the server receives a request from an end user browser.
- Step A:

  The runtime software loads and preprocesses the schema designated in the end user request file.

  The loading and preprocessing may be in a preferred embodiment, identical to the

client-side version, including the optional initialization of the schema or initialization of the values by a custom ASP script, with the following exception:

The schema gets saved to a temporary folder on the server.

The active page is set to be the first page.

- Step B:

  The runtime software applies multiple XSL style sheets in the well-known manner to convert/transform the first page of the schema from XML to HTML. The transformation includes form/page/section headers and help text, as well as information regarding input fields. This includes captions, different HTML controls to capture input, required icons, error messages in case there have been any errors, etc. Also included are two hidden textboxes that contain the globally unique ID assigned to this form instance and the active page number. As noted earlier, this unique ID allows the server to keep track of which client called.

- Step C:

  The server sends the resulting HTML to the browser.

- Step D:

  The browser displays the page. Note: there is minimal or no runtime software on the client, i.e. no JavaScript, no XML nor XSL.

- Step E:

  The user may now fill in the values for the fields and use the navigation buttons.

- Step F:

  Upon activating a navigation button, all of the field values inputted by the end user plus the values from the hidden textbox (globally unique ID and active page number) are sent to the server, as well as which navigation control was chosen by the user.

- Step G:

  The runtime software on the server extracts the unique ID from the received data and loads the saved schema.

- Step H:

  The server runtime software then transfers page values to the designated schema using the ASP Request object to read the incoming data and the XML DOM to write to the schema.

- Step I:

  The server runtime software then validates the received page values against all

validation rules and updates the schema with any error information. The schema is then saved again on the server using a unique ID.

Steps I1 and I2--Page calculation: (Errors? / Last Page?)

If errors occurred, then step I1 causes the same page to be displayed again (with the error messages).

Otherwise, if it is the last page, then step I2 causes the preview to be shown (Step J).

If it is not the last page, then step I2 causes the execution to update the active page number and displays that page (Step B).

Note that the preview is optional. If the preview is disabled for this form instance then the data gets submitted after the last page has been filled and the submit button activated.

- Step J:

  The server runtime software then uses XSL style sheet to transform schema (with data) into HTML for the preview page, as well as adding hidden textboxes for globally unique ID and active page number.

- Step K:

  The server then sends the transformed HTML to the browser.

- Step L:

  Browser displays preview page.

- Step M:

  The user then clicks PREVIOUS or NEXT/SUBMIT. If the user clicked NEXT/SUBMIT, then the browser sends to the server the globally unique ID and active page number in step M.

- If the user did not click NEXT/SUBMIT, then the last data entry page is shown again (step B). Otherwise it goes to final processing (step N).

- Step N:

  The runtime software at the server then loads the saved schema using the globally unique ID. The XML schema is transformed into another (typically) reduced-size XML document using an XSL style sheet for this purpose. As noted previously, this conversion reduces the size of the document by only including needed information, digitally signing the document or transforming into any other desired format. The resulting schema is saved to the message queue. Optionally a post-processing script is

executed. The runtime software then sends an acknowledgement and/or next form/page to the browser.

Process Workflow of eAgent Architecture

Referring to Fig. 6, there is shown the basic system configuration for the eAgent tiered processing architecture of the present invention. It should be noted that this system architecture may be used independently of the forms creation and completion processing discussed earlier. Referring to the figure, one or more message queues 26 communicates with a queue abstraction layer 40. The queue abstraction layer 40 communicates with a queue manager 42, which, in turn, communicates with an eAgent manager 44. The eAgent manager 44 then communicates with an eAgent action coordinator 46. The eAgent action coordinator 46 provides communication among a plurality of eAgents 48-60. Each of these elements 40-46 is a configurable XML COM component. The functions performed by these components will be discussed below. The configuration provides a decoupled eAgent architecture in which action components can be added without having to recompile any code, stop and restart any services, or interrupt functionality in any way.

Fig. 7 provides an execution flowchart for the architecture disclosed in Fig. 6.

- Step A:

  The eAgent manager 44 loads and reads its configuration settings documents 45. The configuration settings on the documents 45 include which queues 26 it should communicate with, where selector files (to be discussed below) are located, how many retries to perform for each XML document, and other initialization information.

- Step B:

  The eAgent manager loads and caches selector XML documents 47 for later use. XML selector documents inform the eAgent Manager which action sets to perform for each XML document that it receives from the Inbox. The selectors use a content based selection using XSL patterns.

- Step C:

  The eAgent manager 44 instructs the Queue Manager 42 to poll an Inbox Queue 35 in the message queues 26 for XML documents. The Queue Manager 42 destructively reads the XML document, and copies the document from the queue to an In-Process Queue 34 and to a log queue 37. It then sends the XML document to the eAgent Manager 44.

- Note that step B and C occur periodically.
- Step D:

The eAgent manager 44 now tests the incoming XML document against each selector document 47 that has been loaded into the eAgent manager 44. A selector document 47 contains one or more XSL selector patterns that are associated with an Actionset ID. The test involves applying the XSL selector pattern(s) to the XML document and determining if any matches were found. If matches for one or more (as configured) XSL selector pattern were found, then an Actionset designated by an Actionset ID associated with the particular selector pattern will be executed (not in this step). . If no selector pattern matches are found, then it continues to step E2. If the eAgent Manager 44 finds one or more Actionset(s) ID's for Actionset(s) to execute, it continues to Step E. In addition, the eAgent Manager 44 creates a routing document, to which it attaches the incoming document. The purpose of the routing document is to provide tracking functionality of actions performed on the XML document, in addition to routing the document itself.

- Step E:

If an Actionset ID is found (i.e., a match to a selector pattern has been determined), then the eAgent Manager 44 instructs the eAgent Action Coordinator 46 to load an Actionset document 49 associated with the Actionset ID associated with the matched selector pattern in step D, and execute the actions specified in this Actionset document 49.

- Step E2:

If the eAgent Manager found no matching selector file for the current XML document, it instructs the Queue Manager 42 to remove the XML document from the In-Process Queue 34, and add it to the Dead-Letter Queue 33.

- Step F:

The Actionset file 49 includes at least one action, with that action containing a ProgID or GUID for a relevant eAgent, plus an action document ID. Based on the Actionset file 49 specified in steps D and E, the eAgent Action Coordinator 46 loads an eAgent (48-60) specified by a ProgID for the Actionset file 49 and instructs the eAgent to perform the relevant task by passing it to the associated document ID.

- Step G:

The loaded eAgent reads the action document associated with the specified action document ID (from the Actionset file 49) (each eAgent has its own type of associated Action document which tells it which internal methods to call for each action specified by the eAgent Action Coordinator 46), and performs the requested tasks.

➢ Step H:

Once the tasks are completed, the eAgent returns any resulting data from its actions to the eAgent Action Coordinator 46. The eAgent coordinator 46 adds information to the routing document. Typically, this information will relate to the eAgent action, and may provide information on the completion or non-completion of the action by the eAgent. The information may also include a time-stamp for the time when the eAgent commenced its action, and a time-stamp for when the eAgent completed its action. This information may include results from the action performed by the eAgent. This information may be readable from the routing document by other eAgents. Note that the time-stamp information provides a time history of activity for tracking purposes.

- If the action was not successful, the eAgent Action Coordinator 46 informs the eAgent Manager 44 that the action was not successful and passes the routing document and the attached XML document to the eAgent Manager. It will also add information relating to this non-completion to the routing document noted above.
- If the action was successful, the eAgent Action Coordinator adds this information to the routing document, and checks if this was the last action specified in the Actionset.
- If it was not the last action, step F is performed again for the next action in the Actionset.
- If it was the last action, then the eAgent Action Coordinator 46 checks if the current Actionset is the last Actionset specified in the original selector file for this XML document.
- If it was not the last Actionset, the eAgent Action Coordinator 46 moves to step E for the next Actionset.
- If this was the last Actionset, it passes the routing document with the included original XML Document back to the eAgent Manager 44. As

noted above, the routing document will include information on whether all the Actionsets for this XML document were successful or not.

- If the Actionsets were not successful, the eAgent Manager performs step I1 and either writes to the error queue 30 or the retry queue 31, depending on the eAgent manager configuration settings, and removes the document from the in-process queue 34.
- If the Actionset(s) were successful, the eAgent Manager 44 performs step I2, i.e., it sends the routing document with the original document and pertinent information attached to a journal queue 32, and deletes the document from the in-process queue 34.
- In addition, as noted above, timestamps are added to the routing document detailing when the action was initiated and when complete.

➢ Step I1:

If the Actionsets were not successful, the eAgent Manager sends the routing document to the Error Queue 30, or to the Retry Queue 31. The configuration settings read in step A specify how many times an action should be retried and how long to wait between each retry. This determines whether the eAgent Manager 44 sends the XML document to the Retry Queue 31 or the Error Queue 30. The eAgent Manager 44 also deletes the routing document from the In-Process Queue 34 since it is no longer in process.

➢ Step I2:

If the Actionsets were successful, the eAgent deletes the XML document form the In-Process Queue, and places it in the Journal Queue, indicating success (and logging the document for auditing purposes).

To summarize the organization, each selector file includes one or more patterns, and will be triggered when one or more of these patterns are matched in the incoming document. The selection of which and how many patterns must be matched is determined by the processing designer. Each selector file includes only one Actionset (although Actionsets can be nested). Each Actionset includes one or more actions. Each action includes the designation of one eAgent ID and an action document ID.

It should be noted that Actionsets may include branching commands, repetition commands, parallel execution commands, and time controlled execution commands.

The present invention also encompasses a system shown in Fig. 20 for performing actions on documents in a computing environment. The system comprises a selector file database 300 containing a plurality of selector files, with each selector file including one or more patterns. The system further comprises an Actionset database 302 including a plurality of Actionsets, with each Actionset including one or more actions that reference an eAgent ID and an action document ID. Each selector file is associated with exactly one Actionset (but there may be nesting of Actionsets). The system further includes an eAgent specific action document database 304 containing a plurality of action documents that are specific to eAgents. Note that each Actionset is associated with at least one action document.

The system may further include at least one creation module for creating and/or manipulating entries in one or more of the databases 300-304.

The system may further comprise a processing component 306 programmed with code for allowing a designer to select at least one of the selector files from the selector file database 300, code for allowing the designer to select at least one of the Actionsets from the Actionset database 302, code for specifying at least one eAgent specific action document from the eAgent specific action document database 304, code for linking the specified selector file from the database 300 with the specified Actionset from the database 302, and code for linking the specified Actionset with the specified action document.

Referring now to Fig. 8, there is shown a system block diagram of an integrated development environment (IDE) architecture in accordance with the present invention. The IDE architecture of Fig. 8 is based on a distributed, n-tier architecture, that allows sharing of both user interface and business components with centralized, dynamic configuration and automatic distribution. This is described in detail in the following sections. Note that the term "tiered architecture" means that a component in a given tier can communicate with other components in the same tier or with components in immediately adjacent tiers only.

Distributed – Accessing Files/XML documents over the Internet

Accessing files/XML documents is performed using a file system abstraction layer 100, an ASP File System client implementation 102 and an ASP File System server implementation 104.

The file system abstraction layer 100 is an interface that defines and contains common file manipulation functions, such as reading, writing and deleting files/documents.

The ASP File System client implementation 102 implements an interface and communication between the file system abstraction layer 100 and the server implementation 104 using its own XML-based simple file access protocol (SFAP). SFAP includes the commands as defined in the file system abstraction layer in an XML format.

The server implementation 104 performs actual file manipulation, e.g. reading a file and communicates with the client-side implementation 102 using the SFAP.

Therefore, the user (i.e., the business components) of the file system abstraction layer 100 can manipulate files/XML documents over the Internet simply by using the functions of the file system abstraction layer.

N-tiers

Presentation Host Tier - IDE host.

An IDE host tier 110 is shown in the figure defines an environment into which different user interface components (IDE components) can be plugged in. The host does not provide any end user functionality itself, but defines a standard set of interfaces for IDE components to implement, including

a) A shared library of user interface and logic elements, including treeview, listview, toolbar controls.

b) A consistent navigation system, regardless of the IDE component.

c) A set of "standard actions," such as new/edit/delete

d) Provision to supply custom controls.

e) Ability to have varying number of IDE components hosted.

Important additional aspects of this configuration include:

- Implemented both standalone and inside a browser.
- Support for standard actions
- Support for sharing business components

Note from the above, that along with the shared user interface and logic code components, the presentation host 110 includes code for reading any IDE component code and business logic component code that may be defined by an IDE configuration 107. The IDE configuration block 107 stores a file for each of a plurality of different roles, with the file for each different role including the ID's and other appropriate information

for the components that are to be loaded into the computing resource or browser that is to perform that given role.

Presentation Controller Tier - IDE components

The IDE components 108 implement the interfaces defined in the IDE host 110 and are the components that actually define the interactions with the end user, and include such IDE Components as eAgent Actions code, Form Schema code, Queries code, and Regular Expressions code. They do not include the business or application logic. The IDE components code is typically loaded to the browser or other computing resource by the presentation host 110. The IDE components 108 also typically have associated custom controls. Custom controls are required for the result pane of the window (right hand side of the IDE) if some other view than the listview is to be shown.

Business Logic Tier - Business Components

The business components 106 implement the business rules/application logic for that component for its associated component 108 in the presentation controller tier. By way of example, the logic for the select query component in the business logic tier might include a selectquery class with a method "checkifvalid" that checks if all necessary information has been provided and all data entered is valid. Likewise, the logic for the form schema component in the business logic tier might include classes for the relationships between forms, pages, sections and fields, where these classes have attributes that describe the objects, such as a caption attribute of the field class.

Data Access Tier

This tier includes code to allow the business component code to read and write information to/from a computing resource. In a preferred embodiment, the tier includes the file system abstraction layer 100, the file system client implementation 102, and the file system server implementation 104 described above.

Data Tier

This tier includes an eAgent repository 105 which includes selector files/Actionset and eAgent specific action document databases, the schema library 10, the query library 12, the validation library (regular expressions) 14, and the IDE configuration 107. These repositories/libraries are where the actual XML documents reside. This could be in the file system or in an XML database.

Sharing of User Interface Components

The IDE host 110 allows the sharing among components 108 of common interface elements, such as toolbar, treeview and listview. In regards to the treeview, the IDE host 110 also creates a "virtual root" for each IDE component, so that each IDE component may only see and manipulate its part of the tree.

Sharing of IDE components allows for a lower footprint, e.g., instead of five treeview controls, only one is used.

Sharing of Business Components

Unlike any other system, the IDE host 110 has the explicit facility of using business components 106 to implement the business logic of the IDE components and the ability to share them between different IDE components. As the exact same objects/instances are shared among different IDE components there are no synchronization delays.

The instancing of business components is therefore not done inside the IDE components but instead from the IDE host.

A configuration XML document that is located in the IDE configuration component 107 defines:

- which IDE and business components are to be loaded by the presentation host 110 loading code;
- which IDE components 108 are associated with which business components 106
- any dependencies between business components;
- other configuration settings for each business component.

Using this information the IDE host 110 creates instances of the business components in the proper order and passes them and the relevant configuration information to the IDE components 108, which in turn pass the relevant information to their business component(s) 106.

Note: the IDE host may be componentized. By way of example, in the preferred embodiment the IDE host is actually made up itself of at least 3 different components; a control component, a communications and a configuration component.

Centralized, dynamic configuration

The configuration is centralized, i.e. it is set by the Web Server or other appropriate computing resource via the IDE configuration block 107 which lists the components that are to be loaded to a given browser or other computing resource to carry out a role. Thus, there is no need to do the configuration on each client machine. This is

because the configuration XML document resides on the Web Server in the IDE configuration block 107.

Note that because of this architecture, the configuration of any given browser or other computing resource is dynamic, i.e., the IDE host 110 does not need to be recompiled if more or less IDE components are to be loaded. The number of IDE (and business) components to load is determined at runtime/startup.

Automatic Distribution

By using an ActiveX based solution running in Internet Explorer 5 or another browser software compatible with a data-centric hierarchical storage and retrieval mechanism, both installing and upgrading is handled automatically. This is handled by Internet Explorer, i.e., IE 5 checks at the beginning whether the components are registered on the machine or if the newest version is on the machine and if not it automatically installs or upgrades the individual components onto the client machine.

To summarize one aspect of the present invention, there is provided an integrated development environment method, comprising the steps of: loading onto a computer resource code in a tiered architecture comprising: a presentation host tier that includes presentation host code, the presentation host code including shared user interface and logic code components, code for reading a IDE configuration, and code for loading any IDE component code and business logic components code as defined in the IDE configuration; an IDE component tier, including at least one IDE component code loaded by the presentation host on the user-interface program, the IDE component code including a user interface; a business logic component tier, including at least one business logic component code loaded by the presentation host on the user-interface program, the business logic component code defining the interrelationships between objects represented in the user interface; a data access abstraction layer tier, including a data access abstraction layer code that allows business components code to read and write information to/from a computing resource; and communicating with a remote computing resource through use of the data access abstraction layer.

Process Workflow Diagram of Form Publishing Process

Referring now to Fig. 5 and Fig. 21, there is shown a workflow diagram for a forms publishing process and a schematic diagram of the IDE – server operation, respectively. With reference to Fig. 5, the process is as follows:

- After saving a new form in execution block 200, the user is prompted in execution block 202 to publish to the web or other location.
- If he chooses yes, then the publishing process is initiated
- Step A:

The publishing wizard starts up and requests a list of templates from the "publish.asp" page. Note: this file can be named differently.

shows an XML selector schema. Fig. 17 shows an Actionset schema. Fig. 18 shows an action Query eAgent schema. Fig. 19 shows a Queue Config schema. The various element/attribute descriptions are listed below.

Note: bold-faced items denote XML elements, non-bold-faced items denote XML attributes.

Element/Attribute	Description
form	The highest level element acting as a container for all of the data items that make up the intended form.
name	Unique identifier for the form. May only contain characters, no spaces or special characters allowed.
caption	Title of the form, may contain spaces and special characters.
helptext	Informative text such as instructions on what this form is about.
page	A subdivision of a form that typically is displayed as a separate page in the browser.
name	Unique identifier for the page. May only contain characters, no spaces or special characters allowed.
caption	Title of the page, may contain spaces and special characters.
helptext	Informative text such as instructions on what this page is about.
section	A subdivision of a page that contains a number of fields for the user to enter.
name	Unique identifier for the section. May only contain characters, no spaces or special characters allowed.
caption	Title of the section, may contain spaces and special characters. If there is more than one section per page, the sections must contain a caption attribute, otherwise it is ignored.
helptext	Informative text such as instructions on what this section is about.
isvalid	true/false - flag reflecting the status of the section: true if the items within the group have no validation errors associated with the item, false otherwise
errormessage	The text that will be displayed if a validation rule has been broken
jscriptvalidation	A validation rule written in JavaScript that is to be evaluated for this section. No direct access to outside resources such as databases, etc. is needed.
jscriptexternalvalidation	A validation rule written in JavaScript that is to be evaluated for this section. Direct access to outside resources such as databases, etc. is needed.
captionrow	true if the section is to be displayed with the captions of all of the first element in each row of items on a separate row above the rest of the items

field	A data entry unit, which may be displayed in different fashions. A field is an entity that accepts user input.
name	Unique identifier for the field. May only contain characters and digits (not as first letter), no spaces or special characters allowed.
caption	Caption of this field, may contain spaces and special characters.
helptext	Informative text such as instructions on what to enter in this field.
isvalid	true/false - flag reflecting the status of the item: true if no validation rules for this field have been violated; false otherwise.
errormessage	Error message being displayed to the user if a validation rule for this field has been violated.
validexample	A valid example for this field.
jscriptvalidation	A validation rule written in JavaScript that is to be evaluated for this field. No direct access to outside resources such as databases, etc., is needed.
jscriptexternalvalidation	A validation rule written in JavaScript that is to be evaluated for this field. Direct access to outside resources such as databases, etc. is needed.
minlength	The minimum number of characters required for a field.
maxlength	The maximum number of characters allowed for the field.
required	False if empty entries into this field are allowed, true otherwise.
minvalue	Minimum value allowed for this field.
maxvalue	Maximum value allowed for this field.
capitalization	Provide for automatic formatting of input, e.g. capitalize all, make all lowercase or put into proper case
datatype	Allowed input format.
regexvalidation	A regular expression pattern that is to be applied to the field value input.
regexmodifier	A modifying value that changes how the regular expression is applied to the field value: 'i' (incasesensitive) will check the input value without regard to case and 'g' (global) will check for all instances of the pattern (as opposed to stopping at the first match).
text	The text node for the field. It contains the default value in the beginning and will contain the entered data from the end user as soon as the user enters data.
display	The display element provides basic information on how to display the field.
samerow	true if the item is to be displayed on the same row

	with the previous field. Not included if the field is the first field of a row.
nowrap	true if the values associated with the display of the field are allowed to wrap to the next line, false otherwise
width	width of the input control in characters.
controltype	Name used to determine how to display the field element that will be used to accept user input.
cols	Number of characters across to be displayed for a textarea display.
rows	Number of characters down to be displayed for a textarea display.
multipleselect	true if multiple checkboxes in a group of checkboxes or selections in a pull-down list may be selected. Not included if only a single value may be chosen.
selectprompt	Text to be shown as the first field in a pull-down list to prompt the user to select an entry.
minyear	Minimum year displayed in datepicker year select
maxyear	Maximum year displayed in datepicker year select
multitextformat	Format specifying how many textboxes to show and relevant properties, such as maxlength.
multi	true if more than one field are to be shown in a row.
rownumber	A number identifying the sequence in which the fields will be displayed down the section.
rowindex	A number identifying the sequence in which the fields will be displayed across the page.
entries	Provides a list of possible values for this field, which may be used for display and validation purposes.
isfromdb	true if a query to a database is required
qdef	ID/Name of the query definition containing the query to return the list of entries
entry	A single entry.
show	The text to be displayed to the end user which may be different from the internal value stored
value	The value of that entry.
parameter	A parameter for use with queries that further specifies it.
name	Name of the parameter. Must correspond to the name of the parameter for the select query
value	The value for this parameter.
custom	An element that allows addition of custom data to any field or section as a name-value pair.
name	The name of the custom entry
value	The value of the custom entry
data	This element is a container for system, temp and

	user data that are filled at runtime.
system	Data needed by the system, such as path settings
guid	A globally unique identifier that is used to identify a specific instance of a form.
temp	Temporary data needed by the system or the user
pagenumber	Number of the current page being displayed
serverpath	URL of the web server containing the application
schemapath	Location of the directory that contains the schema.
xslimagepath	Location of the directory containing the images used by the XSL files
user	Element to place custom data regarding sections.

7. Regular Expression

Element/Attribute	Description
regexdb	Root element, just a container of regex elements
regex	Contains a regular expression
name	The name of the regular expression. May only contain characters, no spaces or special characters allowed. Must be unique.
pattern	The pattern that describes what is looked for.
globalsearch	A regular expression modifier that defines whether the pattern shall apply to all of the string or only part of it
casesensitive	A regular expression modifier that specifies whether to take case sensitivity into account or not.
description	A description of the regular expression that explains what it is about.
validexample	An valid example of an entry.
errortext	Text to be displayed when an entry does not conform to the pattern
regextest	Contains the information necessary to perform tests and quality assurance against the pattern.
testvalue	A value that is to be tested/evaluated against the current regular expression to determine validity.
expectedresult	True when one expects the testvalue to be a valid entry, false when one expects the testvalue to be not valid.

8. Select Query

Element/Attribute	Description
selectquery	The selectquery itself. Root element.
conn	The name of the connection to use. This is mapped to a connection file, specifically to a Microsoft Data Link file with the extension UDL
type	The type of query. This is the same as ADO's CommandTypeEnum. The possible values are:

	1 – Text 2 – Table 4 – Stored Procedure 8 – Unknown 256 - File 512 - TableDirect
commandtimeout	Number of seconds until this command times out
description	description of this select query
parameters	Container of parameter elements
parameter	Constitutes the one parameter that can be passed at runtime to modify the query text.
id	unique parameter name identifying this parameter
description	description of this parameter
datatype	The data type of this parameter
min	The minimum value for this parameter
max	The maximum value for this parameter
example	An example for a value passed to this parameter
autoformat	Specifies whether or not the parameter should be autoformatted, i.e. stripped of leading and trailing white spaces and that single quotes be replaced with 2 single quotes.
textnode	is not an element but is mentioned here because in this case more than one textnode may be used. This contains the actual query text, e.g. "Select * from authors"
refparameter	a reference to a defined parameter, that shall be replaced at runtime with the passed value
id	the id of the parameter

9. Execute Select Query (ValSelect)

Element/Attribute	Description
executeselectquery	The executequery itself. Root element.
location	Full path to the FOLDER where the query definition file resides (Note that the UDL connection file required for the query must also reside there)
name	Name of the query definition file (without the extension)
cursortype	ADO cursor type. Defaults to forward-only (0)
parameter	Each parameter needed for the query is in its own element
name	Name (id) of the parameter
value	Value of the parameter

10. Templates

Element/Attribute	Description

templates	Root element.
template	Element describing a particular template
name	Name of the template, e.g. "SMC Look"
path	Full path to the folder where the template resides, e.g. "c:\templates\smclook"

11. Template

Element/Attribute	Description
template	The template itself. Root element.
parameters	Element containing parameters.
parameter	Element describing one parameter.
id	Attribute that identifies a particular parameter.
question	Attribute containing the question the user is asked to describe this parameter, e.g. "Does this form have a banner logo image?"
type	The type of parameter – text, boolean or list
listitem	If the parameter has a list of items associated with it.
show	The text to be displayed as one entry.
value	The internal value stored
templateitems	This element contains all the template items that are to be copied
templateitem	Each item of the template
id	The identifier for the template item – the name of the file, e.g. "CS.asp"
scanforparameters	Boolean attribute set to true if there are parameters to be replaced by values specified by the user.
keepname	Boolean value specifying whether to change the name of the item.
subfolder	The name of the subfolder where the item resides (if the item is in a subfolder)
allowoverwrite	Boolean value specifying if the destination file should be overwritten if one already exists.

12. Publishfolders

Element/Attribute	Description
publishfolders	The publishfolders root element.
folder	The element describing one publishing folder
name	Descriptive name of the folder e.g. "SMC Internal Forms"
path	Full path to the folder, e.g. "c:\inetpub\wwwroot\smcintforms"
urlrelativetowebserverroot	The url of the publishing destination folder relative to the root of the webserver, for example if the full url to the publishing folder is: http://www.someplace.com/SomeRoot/SomeSub/Ta

	rget The urlrelativetowebserverroot attribute value will be: SomeRoot/SomeSub/Target

13. IDE Configuration

Element/Attribute	Description
ideconfig	The IDE configuration root element.
idecomponents	The root element for all of the IDE components
idecomponent	One single IDE component, which may have one or more business components on which it is dependent on
progid	The program ID of the IDE component, e.g. WTKIDECQueries.CIDEQueries
businesscomponent	Child of idecomponent, referencing a business component as defined under "businesscomponents"
progid	The program ID of the business component.
config	Child of idecomponent. Allows custom subelements which include configuration settings for a IDE component, such as display preferences.
businesscomponents	The root element for the business components
businesscomponent	Child of business component, defines a business component, which may be used by one or more GUI components.
progid	The program ID of the business component.
businesscomponent	A child of business component, indicating a dependant business component.
progid	The program ID of the dependant business component
config	Child of businesscomponent. Allows custom subelements which include configuration settings for a business component, such as path settings.

14. XML Selector

Element/Attribute	Description
selector	Root element that defines, just a container of regex elements
Actionsetid	ID of the Actionset to execute if the required XSL patterns match the XML document.
matchall	If true then all of the XSL patterns have to be successful, if false then only one pattern needs to match.
priority	A number specifying how to deal with the Actionset if the patterns apply. The lower the number the

	greater the priority. Actionsets with higher priorities are executed first.
pattern	Only contains a text node, that contains the XSL pattern that selects one more nodes from the XML document.

15. Actionset

Element/Attribute	Description
Actionset	Root element that contains the references to the actions to execute on an XML document.
action	Single action that is to be executed on an XML document.
progid	The program id of the eAgent that deals with this action.
actionid	The id of the action to execute

16. Action Query

Element/Attribute	Description
actionquery	The actionquery itself. Root element.
conn	The name of the connection to use. This is mapped to a connection file, specifically to a Microsoft Data Link file with the extension .UDL
type	The type of query. This is the same as ADO's CommandTypeEnum. The possible values are: 1 – Text 2 – Table 4 – Stored Procedure 8 – Unknown 256 - File 512 - TableDirect
commandtimeout	Number of seconds until this command times out.
description	description of this select query
parameters	Container of parameter elements
parameter	Constitutes the one parameter that can be passed at runtime to modify the query text.
id	unique parameter name identifying this parameter
description	description of this parameter
datatype	The data type of this parameter
min	The minimum value for this parameter
max	The maximum value for this parameter
example	An example for a value passed to this parameter
autoformat	Specifies whether or not the parameter should be autoformatted, i.e. stripped of leading and trailing white spaces and that single quotes be replaced with 2 single quotes.

statement	Unlike a select query an action query may contain more than one statement, that's why there is an explicit statement element.
textnode	is not an element but is mentioned here because in this case more than one textnode may be used. This contains the actual query text, e.g. "delete * from authors"
refparameter	a reference to a defined parameter, that shall be replaced at runtime with the passed value
id	the id of the parameter

Form Templates--Purpose

The purpose of form templates is to have a separate repository for the look and feel of the HTML pages in which the forms will be displayed.

Sample Folder Structure

```
_mflibrary
|
|
-- templates
   templates.xml
   |
   |
   -- images
   |
   |
   -- css
   |
   |
   -- ssscript
   |
   |
   -- csscript
   |
   |
   -- SMCLook
      Main.asp (required file, must be there and must have that name)
        CS.ASP
        SS.ASP
        template.xml (required file, must be there and must have that name)
      |
      |
      -- SmcSciDisplay
        pagebanner.htm (in the example below, this one gets copied)
        smclogo.gif (in the example below, this one does not get copied, because there is
no <templateitem. for it)
```

Note: SMCLogo.gif could be in the images folder, but that's up to the templates designer

Templates.xml

```
<templates>
    <template name="SMC Look" folder="c:\templates\smclook"/>
    <template name="NIH Look" folder="c:\templates\nihlook"/>
</templates>
```

Template.xml

```
<template>
  <parameters>
    <parameter id="formcontact" question="Please enter the eMail address for the
form contact" type="text"/>
    <parameter id="showpictures" question="Do you wish to show pictures for this
form?" type="boolean"/>
    <parameter id="colorscheme" question="Please select the color scheme you
wish to choose" type="list">
      <listitem name="golden" value="1"/ >
      <listitem name="green" value="2"< / >
    </parameter>
  </parameters>
  <templateitems>
    <templateitem id="CS.ASP" scanforparameters="true" keepname="false"
subfolder="" allowoverwrite="false"/>
    <templateitem id="pagebanner.htm" scanforparameters="false"
subfolder="SmcSciDisplay" keepname="true" allowoverwrite="false"/>
      <templateitem id="pathsettingsjs.asp" scanforparameters="true"
folder="LocalServerSideScript" keepname="true"

allowoverwrite="false"/>
  </templateitems>
</template>
```

Note:
- Main.ASP must exist and will be renamed according to the name chosen by the forms designer/publisher.
- Per the definition used in this context, a templateitem is one that needs to be copied.
- All templateitems must reside inside or below the template folder.

Note:
- The name of the schema must be provided by the publishing wizard (no user input necessary)
- The name of the publishingfolder, where the ASP pages will be copied to, is provided by the publish.ASP page. (the publishingfolder is the one where main.asp and all other relevant items will be copied to)
- name of the template is provided by user (earlier on)

- The name of the resulting ASP page / form (e.g. Leave.ASP) is provided by the end user. It does not however have to be put into the template, because this is always needed.

Publishing Process

The publishing process was previously described in detail in relation to Fig. 5. The following material supplements that description.

Initiation

1. First, the user saves the form. The user will be prompted to start the publishing wizard. If yes is selected then the wizard is started.
2. User selects the form and chooses "publish to web."

List of publishing folders

Per "publish.asp" there shall be one XML file that lists all available publishing folders. It should look as follows:

```
<folders>
        <folder name="General Forms" path="c:\inetpub\wwwroot\generalforms"/>
        <folder name="HR Forms" path="c:\inetpub\wwwroot\hrforms"/>
</folders>
```

Communication between VB wizard and publish.asp

Step 1: Present list of templates to user. User selects one.
Step 2: If there are any parameters, present corresponding questions and let user enter them one by one.
Step 3: If there is more than one publishing folder, present list of publishing folders.
Step 4: Display name of the resulting form. (Default to name of schema). User clicks Publish.
Note: the path to "publish.asp" is set in the config file and can also be called a different name.

Example Operations

1. Get a list of templates:

```
<request>
        <action>GetTemplatesList</action>
</request>

<response>
        <templates>
                <template name="SMC Look"/>
                <template name="NIH Look"/>
        </templates>
</response>
```

Note: the folder attribute should not be sent to the client to hide this information (e.g. use database later on instead of folders)

2. Get parameters for specific template and the list of publishing folders

```
<request>
        <action>GetParametersAndPublishingFolders</action>
        <parameter>NIHLook</parameter>
</request>

<response>
        <parameters>
                <parameter id="formcontact" question="Please enter the eMail address
for the form contact" type="text"/>
                <parameter id="showpictures" question="Do you wish to show pictures
for this form?" type="boolean"/>
        <parameter id="colorscheme" question="Please select the color scheme you
wish to choose" type="list">
                <listitem show="golden"/ >
                <listitem show="red"/ >
                </parameter>
        </parameters>
        <folders>
                <folder name="General Forms"/>
                <folder name="HR Forms"/>
        </folders>
</response>
```

3.

```
<request>
        <action>formexists</action>
        <publishingfolder>General Forms</publishingfolder>
        <formname>Leave</formname>
</request>

if form exists
<response>true</response>

else
<response>false</response>
```

4.

```
<request>
        <action>publish</action>
        <template>NihLook</template>
        <formschemaname>Leave<formschemaname>
        <publishingfolder>General Forms</publishingfolder>
        <formtitle>Leave Application</formtitle>
```

```
<formschemaname>Leave</formschemaname>
<parameters>
        <parameter id="formcontact">Sanjeev@Science-
Management.com</parameter>
        <parameter id=showpictures>true</parameter>
        <parameter id="colorscheme">golden</parameter>
</parameters>
</request>

if successful:
<response/>

else:
<exception id="15">could not access file c:...</exception>
```

Processing

In a preferred embodiment, the processing is implemented right away in a synchronous manner. However, this will not always be the case, e.g., when the publishing needs to be reviewed and approved before it may be published. In this case it would first get passed to the authorizing person, and only when approved would the actual publishing be performed.

To enable easy switching to a disconnected processing, all of the data that is required for the actual processing of the publication is put into one XML document and passed to a "publish" function. This will allow the publishing process to remain identical when there is a switch to asynchronous processing.

The processing that then takes place first scans and then copies the "Main.ASP" file, and then any other templateitems, and creates subfolders as needed.

Each publication is journalled as part of the processing, so that this can be used later for evaluation, tracking and other purposes. The journaling should preferably include all of the information regarding a publication.

Sample ASP page

```
<html>
<%
  blnShowPictures = <replaceme id="showpictures"></replaceme>
  If blnShowPictures Then
%>
      <img.
<%
      Else
%>
```

...
For questions please contact <replaceme id="formcontact"> </replaceme>
</html>

Absolute vs. Relative Paths

The template designer uses an "#include file=.." for template items that get copied (and uses a relative path setting for this), and uses a "#include virtual=.." for files that not to be copied.

Similarly, for images one would use an absolute path if they are to remain in the template folder. However, care has to be taken not to hardwire the actual web server path, as otherwise one would have to change several settings if one wanted to move from one server to another, such as for debugging. Instead, one should use the WEBSERVERPATH variable that is set in pathsettingsjs.asp

An example of an absolute path:

<img src="<%=WEBSERVERPATH%>/_mflibrary/images/logos/nihlogo.gif" alt width="110" height="103" align="top">

An example of a relative path:
<img src="../images/logos/nihlogo.gif" alt width="110" height="103" align="top">

Note: the same applies to other files with a source attribute, such as Script.

The present invention is a unique product, which includes a rich Integrated Development Environment (IDE), and a powerful runtime environment. The invention allows users to create powerful web based applications using, by way of example, a 4GL Rapid Application Development (RAD) environment. The invention is unique in a variety of important aspects which include the following:

- One aspect is a hierarchical graphical user interface (GUI) that focuses on content instead of on presentation, unlike other tools of this kind, which focus on layout and presentation and are therefore far more time consuming. Such other tools are far more time-consuming to maintain for the same reason.
- One aspect of the invention is the ease of creation AND maintenance of applications, and allows for easy reuse of form pages/sections/elements metadata, and presentation information as well as easy maintenance of these items.

- One aspect of the invention focuses on complete separation of tiers in every aspect of the product. This is done to enable independent creation and maintenance of distinctly separate parts of an application. It allows non-technical people to develop the metadata of web based applications, leaving a minimal amount of work for the web developers (templates and regular expressions) and the database developers (database queries).

  For this aspect, each of the following is stored in a different XML document and is accessed from different tools in the IDE. Each can therefore be performed by different people, and can be maintained separately:

  - the database configuration information
  - the database queries
  - the application presentation templates
  - the application metadata
  - the field validation code
  - custom input patterns/masks for composite fields
- The invention does not require the use of a specialized 'viewer' by users of applications developed using the invention. The only 'viewer' required is a standard browser.
- The invention uses two different presentations of the same metadata to support XML-enabled browsers and non-XML-enabled browsers without any additional development effort by users of the program. XML-enabled browsers perform most processing on the client and, due to less roundtrips to the server, have greatly improved response times.
- The invention uses a queue-based n-tier architecture to increase scalability and reliability.
- The invention uses the eAgent Architecture on the back end to perform multiple scalable and customizable actions and work flows using data obtained from the Invention applications.
-

The following lists more details of the variety of new aspects of the present invention.

1) **Integrated Development Environment**

The Inventive Integrated Development Environment (IDE) allows users to quickly create and maintain web-based applications. The IDE uses a Microsoft Management Console-style interface in one embodiment with a tree view representing the form applications, queries, regular expressions and eAgents.

a) **Architecture**

The IDE architecture is unique.

i) The User Interface configuration for individual users is stored in an XML file. This information specifies which parts of the IDE to load at run time (for example, the configuration file can specify that only the Forms designer part of the IDE should be loaded, and leaving the Regular Expression, Queries, and eAgent sections out of the IDE for that user.

ii) Substantially only Form schema is stored in a unique XML file

- Separating these form schema files from template files makes applications far more maintainable.
- The design stores substantially only the metadata/schema in the XML file used to represent the form. The presentation data is separately stored in a combination of CSS (cascading style sheet), XSL, and ASP files. ASP files are used to store the 'templates' of one or more forms, and these templates form the surrounding graphical presentation of the form. To represent the form fields, CSS files and XSL files are used to convert the XML metadata to HTML for presentation. The advantage of this is that it is very easy to change the entire look and feel of a form. This can be done without touching the XML metadata file.

b) **Graphic User Interface**

i) The IDE runs over the web or a network – i.e. users can develop forms remotely.

ii) The IDE determines which components of the IDE should be loaded at runtime. This is based on an XML configuration file. This allows for different versions of the IDE to be created and deployed very easily. It also allows organizations to give some people access to only some aspects of the IDE. Some of these versions may not include parts of the IDE – for example, the following can be included or excluded:

- Forms (the Forms designer)
- Regular Expressions (validation code)

- Queries (database select queries used to populate form lists)
- Actions (eAgent actions such as emailing form data and inserting data into the database
- Action Queries (database action queries such as insert, update and delete) to be used from the Actions/eAgent section

iii) A method is provided to create a form using a hierarchy of icons with attributes.

Each form/application is represented as an icon with attributes. The attributes allow the user to specify the following:

- Where to store the form on the server
- What to name the form
- What caption the form should have
- Which page to submit the form data to.
- Which graphical template to use when displaying the form.
- Help text to appear on the form.

- Each form page is represented as an icon one level below the form icon in the IDE tree view. The page attributes allow the user to specify the following:
  - The name and caption of the page
  - The order of the page (1st in the list, 2nd etc.) with respect to other pages in the form application
  - Help text to appear on this page.
- Each page section is represented as an icon under the page icon. The section attributes allow the user to specify the following:
  - The name and caption of the section
  - The order of the section (1st, 2nd, etc.) in the current page
  - The help text to appear on this section
  - Section level validation.
- Each section element is represented as an icon under the section icon. The section element attributes allow the user to specify the following:
  - the location of the element within the section
  - element's name and caption

- a valid example of what a user can fill in for this element
- the maximum length of this element
- whether this element has a list or not
- a default value for this element
- help text for this element
- whether the element is required (must the user complete this element in order to submit the form?)
- the data type of the element (int, float, etc.)
- The capitalization of the element value (all upper case, all lower case, proper case)
- the minimum and maximum values for this element
- the name of the Regular Expression used to validate user input. Also, Regular Expressions are kept in separate XML metadata/schema files. However, during form creation, the actual regular expression is inserted into the form metadata/schema.
- Simple comparison validation to other fields
- Advanced comparison using ECMAScript (JavaScript)
- if the control used for this form is a list, whether the list comes form the database or not
- if the list comes from the database, the name of the query used.
- database queries are used to populate dynamic pull-down menu choices on the form. The database queries are stored in the metadata/schema that describes the form. Database queries are defined in a separate section of the IDE and stored in separate XML files.
- database queries can be pre-processed (run at design time) – i.e. the results stored in the metadata/schema document that describes the form.
- The control type used for this element
- If the list is not populated from the database, the list values to be used for this list.

- Whether the control used for this element should be displayed on a new row or not.
- The width of the control used to display the element
- Whether the control used should wrap to the next line
- The type of control which should be used
- If the control is a list, whether the list should be multiselect
- If the control is a list, the prompt to use in the list
- If the control is a text area, the number of rows and columns to be used.

iv) A method is provided to publish forms to the web using a wizard which lists predefined templates and allows a user to select a template for the form, as well as a folder location for the form. The basic task of the publishing wizard is to create ASP pages and associated files in a publishing folder that associates/links a form schema with a template. Template files may have custom parameters/questions that are displayed inside the publishing wizard which allows for flexible on-the fly customization of templates just by answering the questions inside the publishing wizard.

v) A method is provided to save, create, test and retrieve regular expressions using XML files for storage and retrieval.

- Regular expressions are represented as icons in the GUI
- Each Regular Expression may be a set of ASCII characters that describes valid inputs
- The IDE helps users create Regular Expressions
- The IDE helps users test Regular Expressions
- The IDE allows users to import Regular Expressions from an outside data file
- This will allow the sales of Regular Expressions on the Web or network and their seamless incorporation into the IDE.

vi) A method is provided to save, create and test database queries

- Database queries may be represented as icons in the GUI
- Each Database Query is a custom SQL query that is able to interact with relational databases.

- Database connections are described separately – allowing the Database Query to be directed at multiple databases. (The database could be MS Access, MS SQL Server, Oracle, or other database – the user does not need to specify beforehand and can switch at any time without affecting the form.)
- This gives users the ability to change the database used for a project without having to change the form metadata/schema at all.

vii) A method is provided to store and retrieve form elements and form sections in a library. For example:

- A Library of Sections and Fields
- A library from which users can drag and drop form sections and form fields into the form they are currently creating
- LINKED sections and fields will retain the properties described in the library – then, to make a universal change to a field common to many forms, one change in the library will be reflected in all forms. This provides the ability to make a change to a section or form that (optionally) automatically changes all related applications in the users' web site.
- COPIED sections and fields will not retain the properties described in the library and are customizable for each form. Subsequent changes in the library (such as specifying a different database source) will not be reflected in all forms, but only forms that are linked to the library. Users can add any section, or field to the library at any time.

2) Invention Runtime Environment

The Invention Runtime Environment is a highly scalable, maintainable, and reliable runtime environment that uses data created using the IDE or by other means to serve sophisticated web based applications to users with both XML-compatible and non-XML-compatible browsers.

a) Runtime Environment Architecture

The Invention runtime environment is unique and includes a variety of aspects including:

i) A method to serve two or more distinct versions of the form/application using only one form/application description file.

- In one aspect, the design automatically creates and maintains two versions of each application using only one form in the IDE. This means that users of the IDE need only maintain one version of the form/application, but:
  - users of XML-compatible browsers are served a quick, efficient, client-side version of the form, which reduces web server load and increases client-side performance. This version includes sending the schema XML file, and any XSL files, CSS files, and JavaScript files for navigation and validation to the browser for processing.
  - Users of non-XML-compatible browsers are served a pure HTML/CSS version of the form pages which processes each page in the form on the server. This version involves processing the XML schema file, and any XSL files, JavaScript files and COM components on the server, and sending html and CSS to the browser.

A method is provided (folder architecture) to store web files for ultimate ease of maintainability and reuse of application. An important aspect is the fact that the final form files can have minimal size (less than 10 lines) because of the fact that they only have the values that differentiate them from other forms and **LINKS** to both a form schema and a template. What makes this possible is the combination of a standardized folder structure with Server-Side Includes (SSI) that use absolute paths. SSI are a programmatic construct of Active Server Pages that allow the inclusion of content from another file into the existing file at the server. The advantage of this approach is the ability to change both the schema and the look and feel independently both at the time of creation and after the form has been published. There is no need to republish the form for these changes take effect; they take immediate effect. Also, there is a n:m relationship between a template and a schema, i.e., one template can be used with

multiple schemas and one schema can be used with multiple templates. This allows for maximum reuse.

A company can have a standardized look and feel defined for all its forms or a subset of them, and easily associate schemas with this template. If the company wishes to change the standardized look and feel, e.g., the name and logo of the company changed, the change has to be done only in the template and takes immediate effect for all the forms that are associated with this template.

Reuse of schemas:

An application service provider (ASP), in one aspect, may develop and sell a set of standardized schemas for work processes. Several companies/customers may wish to use these standardized schemas, however they may wish to have their own look and feel with the schemas. The ASP provides customized templates for each customer, and associates the resulting forms with the customized templates and the standard schemas. If there is a need to change the schema, e.g., another field is required, then the ASP only needs to change the standardized schema and the change takes immediate effect for all forms that link to this standardized schema.

eAgent Architecture

A further aspect of the invention is the eAgent architecture. It uses a set of COM components and configurable XML files in a preferred embodiment to determine appropriate actions, and to perform these actions for each XML document received in the queue system. Some of the important aspects include:

i) Pattern-content based selection of actions to be performed – an eAgent manager determines which Actionset to perform by searching each incoming XML document for previously cached XSL patterns. Using XSL patterns, a component (the eAgent Manager) determines the type of each incoming XML document, and determines which Actionset to perform based on the type of XML document involved. The component knows which patterns to search for based on the cached XML selector documents.

ii) A hot-pluggable eAgent component architecture: An architecture in which 'action' components can be added without having to recompile any code, stop and restart any services, or interrupt the functionality in any way. More eAgents (with corresponding action files) can be added to the system at any time without interruption. All that is

needed is that any Actionset file which will utilize the new eAgent be updated with the new ProgID of the eAgent, and which actions to perform. Since Actionset files are XML documents, this does not require the stopping and starting or recompiling of any code or code components.

iii) A business rules system which is changeable at run-time (without interrupting normal operation) – the eAgent architecture includes several components that implement the system business rules or work flow. These are the eAgent Manager, the eAgent Action Coordinator, and the eAgents, in a tiered architecture. However, the business rules themselves are defined in XML documents which can be changed without recompiling any code, and in fact can be changed during run-time. (Prior art systems typically would at the very least require the system to be stopped and restarted, and in many cases a complete recompile of the code would be required).

iv) A decoupled action set system: The eAgents in the system are self aware only. They are not aware of the existence in the system of other eAgents. Despite this, actions can be performed which transfer information from one eAgent to another. This is done as follows:

> The eAgent Action Coordinator calls an eAgent function and passes the eAgent a copy of the routing document, which incorporates both the incoming XML document and the results of execution from previous eAgents. It receives back the result of the processing as an XML document/element, which it adds to a routing document attached to the original XML document, before passing the original XML document and the routing document to the next eAgent specified in the Actionset. Another advantage of this is that there is no possibility for an eAgent to corrupt the original message. But more importantly, eAgents need not be designed to work in tandem with any other code or components other than the eAgent Action Coordinator.

Specific eAgents

The current implementation of the invention employs several eAgents:

Action-Query eAgent:

The Action-Query eAgent allows the system to insert, update and delete

data in any ODBC and/or OLE DB compatible database. The data from the incoming XML document can be mapped to fields in the database tables. The mapping is defined in Action-Query documents and the eAgent uses this and the incoming XML document to insert, update and/or delete data in the database.

Schema eAgent:

The schema eAgent allows the system to automatically create and update database tables and action queries based on the incoming XML document. Using this automation facility radically simplifies the development of XML to database mapping.

PDF eAgent:

The PDF eAgent populates an empty PDF form template with data from the incoming XML document. The PDF eAgent action document defines the template to use and other relevant parameters. A naming convention is used to map the information from the incoming XML document to the fields inside the PDF form template.

eMail eAgent:

The eMail eAgent allows the system to send eMails to multiple recipients with a subject, a body, and zero or more attachments. The eMail eAgent action document constitutes the template for the eMail, and is used in coordination with the incoming XML document to construct the eMail.

Script eAgent:

The script eAgent allows an ActiveScript, such as one written in VBScript, JavaScript or Perl, to perform any action on the incoming XML document. As these scripts have access to COM components on the system, including the database, file system, eMail and other components, these scripts offer an enormous amount of flexibility. The access to the incoming XML document is simplified by providing an object model for the script to use.

Sample workflow (simplified):

Business Task

The time for processing of "request for leave" forms must be improved. The HR manager wants to be immediately notified of new requests for leave. S/he also wants a database of all requests so s/he can view leave-related reports. In addition, due to company policies, s/he also needs the requests in writing, signed by the person requesting the leave. Ideally, the data should only be entered once.

Technical Implementation

The form "Request For Leave" is filled out in the browser by the person requesting the leave. When the user submits his/her request the request for leave application (XML document) is submitted to a queue. The queue manager 42 picks it up from the queue and passes it on to the eAgent manager 44, which determines the corresponding Actionset. The Actionset encompasses four actions:

The first action is passed to the Action-Query eAgent with the document ID of "insertLeave". The Action-Query eAgent reads the action document and then inserts the relevant data from the XML document into the database and returns execution to the eAgent Action Coordinator 46, indicating that it has successfully completed it's task. In this way, all requests are recorded in a database which can be utilized for analysis and reporting purposes.

The eAgent Action Coordinator 46 then calls the PDF eAgent with the "leave" action ID. The PDF eAgent reads the action document, finds the location and parameters of the PDF form template and using a naming convention, fills all fields in the template with the corresponding parts of the XML document and saves the resulting PDF document. In addition to returning a success code, it documents the location of this PDF document in the routing document for later use.

The third action involves sending the person who submitted the leave request an eMail with the PDF document as an attachment that was created in the previous action. For this the eMail eAgent is called with the "leaveAck" action document ID. The eMail eAgent loads this action document and constructs an eMail, which contains the person who submitted the leave request as a recipient and contains the PDF form as an attachment. The eMail eAgent then sends this eMail to the person requesting the leave. To fulfill the current company regulations, that person can then print out the PDF form, sign it and pass it on to the HR department. There is no need to enter the data a second time. The eAgent returns a success code to the eAgent Action coordinator 46.

The fourth action involves sending the HR manager an eMail. The eMail eAgent is called by the eAgent Action Coordinator 46 with the "leaveHR" action document ID. The eMail eAgent reads the action document and constructs and sends the eMail to the HR manager.

Additional e-commerce aspects

Faced with today's new, rapidly expanding marketplace, e-commerce merchants have found themselves unable to efficiently and effectively communicate their orders to suppliers and unable to obtain vital business process information, such as real-time inventory levels. Up until now, it has been extremely difficult for companies to conduct business over the Internet because of the lack of a single technical vocabulary for describing business processes.

The only widely adopted e-commerce transaction solution today, Electronic Data Interchange (EDI), has fostered a fragmented market, where trading partners have created "islands of standardization" which support only specific vendors and implementations. Each interconnection must be carefully mapped from a source system to the receiving system and maintained *individually* as source and receiving systems evolve. Each new trading partner adds to the staggering number of unique combinations which must be individually structured and maintained—resulting in global technical gridlock.

EDI's high costs, low ROI and limited services have restricted its market penetration to less than 2% of all US businesses.

Online Transaction Clearinghouse Solution

The eAgent tiered configuration, described as one aspect of the present Invention, possesses the unique capability to provide an unlimited volume and variety of lowest-cost transactions between any vendor and any supplier, through an online transaction clearinghouse.

Based on open Internet technologies and standards, including XML, this aspect of the present invention fulfills a specific niche that has not, until now, been successfully addressed. Using multiple, geographically-dispersed high-capacity servers, this aspect of the invention allows the receipt and processing of intelligent business-to-business (b2b) transaction and workflow instructions, including purchase orders, payment and fulfillment instructions, and price and inventory requests.

This aspect of the present invention facilitates seamlessly integrating all transactions, and will guarantee the distribution of correct transaction instructions, regardless of the ERP systems, legacy systems, or databases used by suppliers and merchants involved in the transaction. The online transaction clearinghouse facilitated by the present invention is a complete supply-side transaction solution designed to maximize accessibility, reliability, responsive service and speed of fulfillment.

Each transaction in accordance with this aspect of the present invention relies on the following services, each of which may be contained within the Forms Web site:

1. **The Guaranteed Transaction.** E-commerce Web sites and/or Forms web sites will send secure, XML-formatted transactions to a global array of Message Queue (MQ) Servers, all poised to capture each e-commerce transaction and forward it to the closest facility. If one MQ Server goes off-line, any of the others will step in to seamlessly take its place.

2. **The Business Rules Package.** The E-commerce Web site' transaction instructions and Forms environment will contain business rules, in addition to the basic transaction that will be executed using the described eAgent workflow engine.

The eAgent.

This aspect of the Invention will use the eAgent architecture to transfer orders and data requests (such as status of order) to a merchant's back-end system and receive information, as appropriate.

This e-commerce aspect of the present Invention is the only solution to fully address all four major sources of transaction costs:

1. The asynchronous systems architecture and intelligent eAgent technology of this e-commerce aspect of the invention together provide an unlimited capacity to receive, store and process information. This unique combination of capabilities engenders near-zero marginal quantitative and qualitative costs of new transactions.
2. This e-commerce aspect of the Invention assures transaction reliability, on-time payment and low charge-back rates.
3. This e-commerce aspect of the present invention provides timely market intelligence and an online adjustable 'smart' business logic vehicle, which combine to enable instant reactions to market fluctuations.
4. This e-commerce aspect of the present Invention drives transaction services and buyer-supplier relationships toward perfect competition, continually reducing the 'asset specificity' portion of the transaction cost.

This e-commerce aspect of the Invention enables enterprises to create a universally accessible value web of increasingly complex, interlocking products and cross-selling arrangements.

From the above, it can be seen that important aspects of the present invention include:

- Use of a **tiered software architecture** that does not compromise on scalability, maintainability and performance.
- **Ease of integration:** The present invention offers exceptional ability to integrate the submitted form data with any other computer system or user by having schema and captured data as an XML document.
- Uses one **metadata** description of forms to display multiple outputs (client-side DHTML/XML/XSL, or server-side Active Server Pages (ASP), or many other possibilities.)
- **Simplicity.** A forms designer needs to only be able to use one simple graphical user interface to create, administer and publish powerful forms to the Intranet/Internet without the need to have any know-how in the areas of HTML/ASP/XML/XSL/CSS/Regular Expressions/JavaScript etc.

Using this innovative approach, the present Invention can be developed and deployed by non-programmers in a short period of time, and over the Internet or over a

network. Changes to forms (maintenance) can be made even more quickly than form creation (again over the Internet). The present invention supports all of the major browsers and is highly scalable, fast and maintainable. And, by having the submitted data available as XML, the results can be easily integrated into any other system. Before the present invention, developing online forms with fields, validation, a preview page, and separate versions for IE5.0 and Netscape would have taken a fully qualified programmer several weeks. Maintaining such an application would have taken the same programmer far longer.

The foregoing description of the preferred embodiment of the Invention has been presented for purpose of illustration and description. It is not intended to be exhaustive or to limit the Invention to the precise form disclosed, and modifications and variations are possible in light of the above teachings or may be acquired from practice of the invention. The embodiments were chosen in order to explain the principles of the Invention and its practical applications in order to enable one skilled in the art to utilize the Invention in various embodiments and with various modifications as are suited to the particular use contemplated. It is intended that the scope of the Invention be defined by the claims appended hereto, and their equivalents.

WE CLAIM:

1. An integrated development method for creating forms, comprising the steps of:

storing at least one schema for a form in a first file at a computing resource, wherein said schema comprises minimal user interface representations;

accessing at least one template of a form in at least one second file separate from said first file, wherein each of said templates comprises a user interface representation of said form;

selecting one of said at least one template and one of said at least one schema; and

creating at least one linking file that links said selected schema and said selected template.

2. A method as defined in claim 1, wherein said schema storing step comprises the step of storing a plurality of different schema.

3. A method as defined in claim 1, wherein a plurality of different templates can be accessed in said template accessing step.

4. A method as defined in claim 1, wherein said schema storing step comprises the step of storing a plurality of different schema.

5. A method as defined in claim 1, wherein said schema storing step comprises creating said schema, including the step of copying at least one element from a different schema into said schema being created.

6. A method as defined in claim 1, wherein said schema storing step comprises the step of copying at least one element from a database of reusable elements to form said schema.

7. A method as defined in claim 1, wherein said schema storing step comprises the step of linking at least one element into said schema from a database of reusable elements.

8. A method as defined in claim 7, wherein said reusable elements are fields.

9. A method as defined in claim 7 wherein said reusable elements is a plurality of said fields.

10. A method as defined in claim 7, wherein said reusable elements are queries.

11. A method as defined in claim 1, further comprising the step of

creating said schema, including the steps of selecting query information from a query information database separate from said first and second files, and incorporating said query information into said schema.

12. A method as defined in claim 11, wherein said query information is linked to said schema.

13. A method as defined in claim 1, further comprising the step of:

creating said schema, including the steps of selecting a regular expression from a regular expression database separate from said first and second files, and incorporating said regular expression into said schema so as to validate inputted data.

14. A method as defined in claim 13, wherein said selected regular expression is linked into said schema.

15. A method as defined in claim 1, further comprising the step of creating said schema, including the steps of selecting a user-defined control from a custom database separate from said first and second files, incorporating said user-defined control into said schema so as to provide improved input for fields made up of a plurality of variable and fixed text elements.

16. A method as defined in claim 1, wherein said schema storing step comprises the step of storing said schema in a data-centric hierarchical storage and retrieval mechanism.

17. A method as defined in claim 16, wherein said data-centric hierarchical storage and retrieval mechanism is XML.

18. A method as defined in claim 1, further comprising the steps of at runtime converting said schema to an interface with a user.

19. A method as defined in claim 18, wherein said converting step comprises converting the schema to HTML.

20. A method as defined in claim 15, wherein said schema includes elements with attributes.

21. A method as defined in claim 20, further comprising the step at runtime of using attribute based validation to validate end user input data.

22. A method as defined in claim 21, further comprising the step at runtime if an error is discovered during said validation step, including an error flag and an associated error message in said schema or separate schema in said data-centric hierarchical storage and retrieval mechanism.

23. A method as defined in claim 22, wherein said error message is presented to a user.

24. A method as defined in claim 1, further comprising the step of: allowing the separate creation and modification of parts of the forms creation and maintenance process and thereby allowing specialization of tasks.

25. A method as defined in claim 24, wherein said tasks include designing queries, regular expressions, user-defined controls, templates and forms creation and processing of the submitted of data.

26. A method as defined in claim 1, wherein said schema comprises at least one field; and

further comprising a field library separate from said first and second files for storing one or more fields; and

wherein said schema storing step comprises the steps of:

storing in a file a schema comprising a link to said at least one field in said field library; and

storing in a file a transformed schema that does not include said link to said at least one field, but rather contents of that field

27. A method as defined in claim 26, wherein said schema includes a plurality of said fields, and wherein said schema includes links to at least two different fields in said field library.

28. A method as defined in claim 26, wherein for a field that is changed in said field library, updating a plurality of said schema that contain a link to said changed field.

29. A method as defined in claim 28, wherein said updating step includes the steps of retransforming said schema.

30. A method as defined in claim 28, wherein said updating step comprises performing said updating step during runtime.

31. A method as defined in claim 1, wherein said linking file creating step comprises the steps of creating a linking file for a data-centric hierarchical storage and retrieval mechanism user-interface program, and creating a linking file for a non-data centric hierarchical storage and retrieval mechanism enabled user-interface program.

32. A method as defined in claim 1, wherein a navigation engine can navigate among a plurality of sections, and wherein said schema storing step comprises a step of storing a schema with at least two sections.

33. A method as defined in claim 1, further comprising the step of defining template specific customization questions.

34. A method as defined in claim 1, wherein said schema comprises at least one field; and further comprising the step of creating a schema and storing a schema including a link from said one field to a field in said field library of reusable fields; and storing a transformed schema with no links.

35. A method as defined in claim 1, if a user interface program is not compatible with a data-centric hierarchical storage and retrieval mechanism, then creating a unique identifier at runtime for an instance of one said storage schemas.

36. A method as defined in claim 35, wherein said unique identifier is passed between a navigation engine in said computing resource to and from a user interface program at a user location.

37. A method as defined in claim 1, wherein at runtime submitted end user data is written to a queue.

38. A method as defined in claim 37, wherein a log of transactions is created for said queue.

39. A method as defined in claim 1, wherein for a user interface program that is compatible with a data centric hierarchical storage and retrieval mechanism, at runtime selecting a schema and sending said selected schema to said user interface program as an island of data in accordance with said data-centric hierarchical storage and retrieval mechanism, within a form page in presentation language.

40. A method as defined in claim 39, wherein said selected schema is in the form of an XML island in an HTML page.

41. A method to publish forms comprising the steps of:

creating a schema;

providing a list of predefined templates;

selecting a template;

selecting a publishing destination; and

linking said schema to said selected template.

42. A method as defined in claim 41, further comprising the step of responding to template specific customization questions and linking responses to said selected template.

43. A method for creating a regular expression file comprising the steps of:

inputting and presenting a name for a regular expression;

inputting and presenting a pattern for a regular expression that is to be used in a validation engine;

inputting and presenting an error message for user inputs that do not match said pattern;

inputting at least one test case; and

comparing said test case to said pattern; and

presenting a message in response to said comparison step.

44. A method as defined in claim 43, further including the step of inputting and presenting a description of said regular expression.

45. A method as defined in claim 43, further comprising the step of inputting and presenting at least one valid example of said pattern.

46. A navigation engine program product, comprising:

a computer usable medium having computer readable program code embodied therein comprising:

first code for transforming data-centric hierarchical storage and retrieval mechanism data to presentation language;

second code for receiving data in presentation language and converting said received data to said data-centric hierarchical storage and retrieval mechanism;

third code for performing attribute based validation on said data;

fourth code for providing navigation among present pages; and

fifth code for saving said data in response to a command.

47. A computer program product as defined in claim 46, wherein said third code operates to perform said attribute validation only after said data has been converted to said data-centric hierarchical storage and retrieval mechanism by said second code.

48. A method for processing forms on a user-interface program, comprising the steps of

receiving an engine and a file with a form including a schema and a template in data-centric hierarchical storage and retrieval mechanism;

transforming with said engine said form from said data-centric hierarchical storage and retrieval mechanism into presentation language and presenting said form at said user-interface program;

receiving user input data in said presentation language;

converting said user input data to said data-centric hierarchical storage and retrieval mechanism;

performing attribute based validation on said converted user input data;

incorporating any error detected in said validation step in said data-centric hierarchical storage and retrieval mechanism into said form schema or separate form schema;

converting said any error into presentation language and presenting an error message associated therewith to said user;

allowing said user to correct said user input data, if necessary; and

saving said user input data.

49. A method as defined in claim 48, wherein said form transforming step comprises the step of receiving a schema as an island in the format of a data centric hierarchical storage and retrieval mechanism in a form page represented in presentation language.

50. A method as defined in claim 49, wherein said schema is stored as an XML island in an HTML page.

51. A method as defined in 48, further comprising the step of presenting a preview of said form to said user with said user input data incorporated therein.

52. A method as defined in claim 48, further comprising the step of allowing said user to navigate among different sections in said form.

53. A method as defined in claim 48, further comprising the step of submitting user data and writing said submitted user data to a queue.

54. A method as defined in claim 53, further comprising the step of creating a log of transactions of said schema.

55. A runtime environment method for an electronic forms applications, comprising the steps of:

receiving a forms request from a user-interface program;

determining if said user-interface program is compatible or not compatible with a data-centric hierarchical storage and retrieval mechanism;

sending from a computing resource to said user-interface program, if said user-interface program is compatible with said language, a schema file containing form information to be inserted into a form template and a form navigation engine and a form input validation engine; and

if said user-interface program is not compatible with said storage and retrieval mechanism, sending user input information to said computing resource for processing and

sending a presentation language version of form pages resulting from said computing resource processing back to said user-interface program.

56. A method as defined in claim 55, wherein at runtime the step is performed of using attributes to validate end user input data.

57. A method as defined in claim 55, where at runtime the step is performed of writing results of said validation and any error message associated therewith into said schema.

58. A method as defined in claim 55, if a user interface program is not compatible with said data-centric hierarchical storage and retrieval mechanism, then creating a unique identifier at runtime for forms processed at said computing resource.

59. A method as defined in claim 55, wherein said unique identifier is passed between a navigation engine in said computing resource to and from said user interface program at a user location.

60. A method as defined in claim 55, wherein at runtime submitted end user data is written to a queue.

61. A method as defined in claim 60, wherein a log of transactions is created for said queue.

62. A method as defined in claim 55, wherein for a user interface program that is compatible with a data centric hierarchical storage and retrieval mechanism, at runtime selecting a schema and sending said selected schema to said user interface program as an island of data in accordance with said data-centric hierarchical storage and retrieval mechanism, within a form page in presentation language.

63. A method as defined in claim 62, wherein said selected schema is in the form of an XML island in an HTML page.

64. A method for creating a form file comprising the steps of:

storing in a form file a path to a schema in a first file at a storage location;

storing in said form file a path to a template in a second file different from said first file at a storage location; and

storing said form file at a user location.

65. A method as defined in claim 64, wherein said step of storing said form file includes the step of storing customization information for a template.

66. A method as defined in claim 64, wherein storing said form file step includes the step of storing customization information for a form.

67. A method as defined in claim 64, wherein said stored paths are absolute paths.

68. A method for performing customized actions in a computing environment, comprising the steps of:

providing at least one selector file, each selector file containing at least one selector pattern and associated with an actionset that contains at least one action, wherein said at least one action contains at least an eAgent ID and an associated action document ID;

obtaining a document for processing from a queue;

searching said document for a match to at least one of said selector patterns of said selector files;

if a match is found for one of said selector patterns, executing said at least one action in said Actionset associated with said matched selector file, including loading an eAgent specified by said eAgent ID; and

said eAgent reading an action document specified by said associated action document ID and performing an action specified therein.

69. A method as defined in claim 68, further comprising the step of obtaining a document for processing from one of a plurality of said queues.

70. A method as defined in claim 68, further comprising the steps of: creating a routing document; attaching said document to said routing document; and attaching information to said routing document.

71. A method as defined in claim 70, wherein said information attached to said routing document relates to a status of the action.

72. A method as defined in claim 70, wherein said information attached to said routing document comprises result information from said action performed by said eAgent.

73. A method as defined in claim 72, wherein said information is readable by a different eAgent.

74. A method as defined in claim 70, wherein said information attaching step comprises attaching said information only from non-eAgents.

75. A method as defined in claim 68, wherein said document is in a format of a data-centric hierarchical storage and retrieval mechanism.

76. A method as defined in claim 68, wherein said document is in XML.

77. A method as defined in claim 68, wherein said selector file, routing document, Actionset and action documents are in a format of a data-centric hierarchical storage and retrieval mechanism.

78. A method as defined in claim 68, wherein said selector file, routing document, Actionset and action documents are in XML.

79. A method as defined in claim 70, wherein said information attached to said document comprises an indication of status of said action by said eAgent.

80. A method as defined in claim 68, further comprising the step of creating a completion file, wherein said routing document is stored in the format of a data-centric hierarchical storage and retrieval mechanism.

81. A method as defined in claim 80, wherein said completion file is stored to a queue.

82. A method as defined in claim 70, wherein said information attached to said routing document comprises a time stamp of at least when the action was completed.

83. A method as defined in claim 70, further comprising the step of determining if a second action is present with an associated second document ID and a second eAgent ID;

said eAgent associated with said second eAgent ID reading an action document specified by said second action document ID and performing an action specified therein;

attaching information to said routing document; and

sending said routing document with said attached information to said completion file.

84. A method as defined in claim 70, further comprising the step of determining if a match to a second selector pattern in said selector files is found, and if said match to said second selector file is found, executing a second Actionset associated with said matched second selector file, including loading an eAgent specified by an eAgent ID in said second Actionset;

said eAgent reading an action document associated with an action document ID in said second action set and performing an action specified therein; and

attaching information to said document.

85. A method as defined in claim 68, wherein said Actionset contains another Actionset.

86. A method as defined in claim 68, wherein said Actionset contains at least one command from a set of commands comprising branching and repetition commands.

87. A method as defined in claim 68, wherein said Actionset contains a parallel execution command.

88. A method as defined in claim 68, wherein said Actionset contains a time controlled execution command.

89. A method as defined in claim 68, wherein said providing step includes providing a plurality of selector files, with each of said selector files containing a priority relative to other selector files.

90. A system for generating forms in a computing environment, comprising:

a schema database;

a template database separate from said schema database; and

at least one linkage file, each containing code to link a selected schema with a selected template.

91. A system as defined in claim 90, further comprising : a database of validation patterns separate from said schema database and said template database; and a database of query information separate from said schema database and said template database

92. A system as defined in claim 91, further comprising a plurality of different forms creation modules, with each of said modules providing a different method of manipulating one or more of said databases.

93. A system as defined in claim 91, further comprising a forms designer configuration database listing which of said modules are to be used by different forms designers at runtime.

94. A system as defined in claim 92, wherein said plurality of modules includes a forms creation modules for allowing a forms designer to create a schema with validation.

95. A system as defined in claim 92 wherein said plurality of modules includes a forms creation module for allowing a forms designer to select regular expressions from said regular expression database for use in said validation.

96. A system as defined in claim 94, wherein said forms creation modules includes code to provide a link in said schema to said selected regular expression in said regular expression database.

97. A system as defined in claim 92, wherein plurality of modules includes:

a forms creation module for allowing a forms designer to select query information from said query information database to create elements in said schema.

98. A system as defined in claim 97, wherein said query selecting forms creation module includes code to provide a link in a schema to said selected query information in said query database.

99. A system as defined in claim 97, wherein said plurality of modules includes a regular expressions module for allowing said forms designer to add/change regular expressions in said regular expressions database.

100. A system as defined in claim 92, wherein said plurality of different forms creation modules includes a query information module for allowing said forms designer to add/change query information in said query information database.

101. A system as defined in claim 90, further comprising an elements database for storing a plurality of different elements that may be reused in forms.

102. A system as defined in claim 92, further comprising an actions module for including code for controlling at least one eAgent.

103. A system as defined in claim 92, further comprising a custom module for creation of user-defined controls.

104. A system as defined in claim 92, wherein said plurality of modules includes a forms creation modules for allowing a forms designer to select a user-defined control.

105. A system for performing actions on documents in a computing environment, comprising:

a selector file database containing a plurality of selector files;

an Actionset database containing a plurality of Actionsets;

wherein each selector file is associated with exactly one Actionset;

an eAgent specific action document database containing a plurality of action documents that are specific to eAgents;

wherein each of said Actionsets is associated with at least one action document.

106. A system as defined in claim 105, further comprising at least one creation module for creating and/or manipulating entries in one of said databases.

107. A system as defined in claim 105, further comprising:

code for specifying at least one of said selector files from said selector file database;

code for specifying at least one of said Actionsets from said Actionset database;

code for specifying at least one of said eAgent specific action documents of said eAgent specific action documents;

code for linking said specified selector file with said specified Actionset; and

code for linking said specified Actionset with said specified action document.

108. A computer program product for an integrated development environment, comprising:

a computer usable medium having a computer readable code embodied therein, comprising:

presentation host code, said presentation host code including shared user interface and logic code components, code for reading a IDE configuration, and code for loading any IDE component code and business logic components code as defined in said IDE configuration;

at least one IDE component code loaded by said presentation host on said user-interface program, said IDE component code including a user interface; and

at least one business logic component code loaded by said presentation host on said user-interface program, said business logic component code defining the interrelationships between objects represented in said user interface.

109. A computer program product as defined in 108, further including a data access abstraction layer code that allows business components code to read and write information to/from a computing resource.

110. A computer program product as defined in claim 108, wherein said presentation host code runs within a browser.

111. A computer program product as defined in claim 109, wherein said data access layer access allows reading and writing information to/from a computing resource over a communications link.

112. A computer program product as defined in claim 109, wherein said data access layer access allows reading and writing information to/from a local computing resource.

113. A computer program product as defined in claim 108, wherein said presentation host includes code for sharing business components code.

114. A computer program product as defined in claim 108, wherein presentation host code, said IDE component code, said business component code, said data access

abstraction layer code are each on a different tier, wherein a tier permits communication only with components in the same or neighboring tiers.

115. An integrated development environment method, comprising the steps of:

loading onto a computer resource code in a tiered architecture comprising:

a presentation host tier that includes presentation host code, said presentation host code including shared user interface and logic code components, code for reading a IDE configuration, and code for loading any IDE component code and business logic components code as defined in said IDE configuration;

an IDE component tier, including at least one IDE component code loaded by said presentation host on said user-interface program, said IDE component code including a user interface;

a business logic component tier, including at least one business logic component code loaded by said presentation host on said user-interface program, said business logic component code defining the interrelationships between objects represented in said user interface;

a data access abstraction layer tier, including a data access abstraction layer code that allows business components code to read and write information to/from a computing resource; and

communicating with a remote computing resource through use of said data access abstraction layer code.

116. A method as defined in claim 115, wherein said communicating step includes the step of communicating over the Internet.

117. A method as defined in claim 115, wherein said communicating step comprises the step of communicating with a remote data access tier in said computing resource, wherein said remote data access tier is a different tier from a data tier in said remote computing resource.

118. A method as defined in claim 116, wherein said communicating step comprises the step of communicating with a remote data access tier in said computing resource, wherein said remote data access tier is a different tier from a data tier in said remote computing resource.

119. A computer program product comprising:

a medium with machine-readable code stored thereon which gives the ability to its computer to perform form creation tasks, comprising:

first code for facilitating a schema database;

second code for facilitating a template database separate from said schema database;

third code for facilitating at least one linkage file, each containing code to link a selected schema in said schema database with a selected template in said template database;

fourth code for facilitating a database of validation patterns separate from said schema database and said template database; and

fifth code for facilitating a database of query information separate from said schema database and said template database.

120. A computer program product as defined in claim 119, further comprising:

a selector file database containing a plurality of selector files;

an Actionset database containing a plurality of Actionsets;

one or more eAgent specific action document databases containing eAgent specific action documents;

at least one module for creation and manipulation of said databases.

121. A computer program product as defined in claim 119, further comprising code for a plurality of different forms creation modules, with each of said modules providing a different method of manipulating one or more of said databases; and a forms designer configuration database listing which of said modules that are to be downloaded to different forms designers at runtime.

122. A computer program product as defined in claim 119, wherein said code for said plurality of modules includes:

code for a forms creation modules for allowing a forms designer to create a schema with validation, and to select regular expressions from said regular expression database for use in said validation.

123. A computer program product as defined in claim 122, wherein said code for said forms creation modules includes code to provide a link in said schema to said selected regular expression in said regular expression database.

124. A computer program product as defined in claim 121, wherein said code for said plurality of modules includes:

code for a forms creation module for allowing a forms designer to select query information from said query information database to create elements in said schema.

125. A computer program product as defined in claim 122, wherein said code for said forms creation module includes code to provide a link in a schema to said selected query information in said query database.

126. A computer program product as defined in claim 121, wherein said code for said plurality of modules includes a regular expressions module for allowing said forms designer to add/change regular expressions in said regular expressions database.

127. A computer program product as defined in claim 121, wherein said code for said plurality of modules includes a query information module for allowing said forms designer to add/change query information in said query information database.

128. A computer program product as defined in claim 119, further comprising code for facilitating an elements database for storing a plurality of different elements that may be reused in forms.

129. A computer program product as defined in claim 119, further comprising code for facilitating an actions module for including code for controlling at least one eAgent.

130. A computer program product as defined in claim 121, further comprising code for facilitating a custom module for creation of user-defined controls.

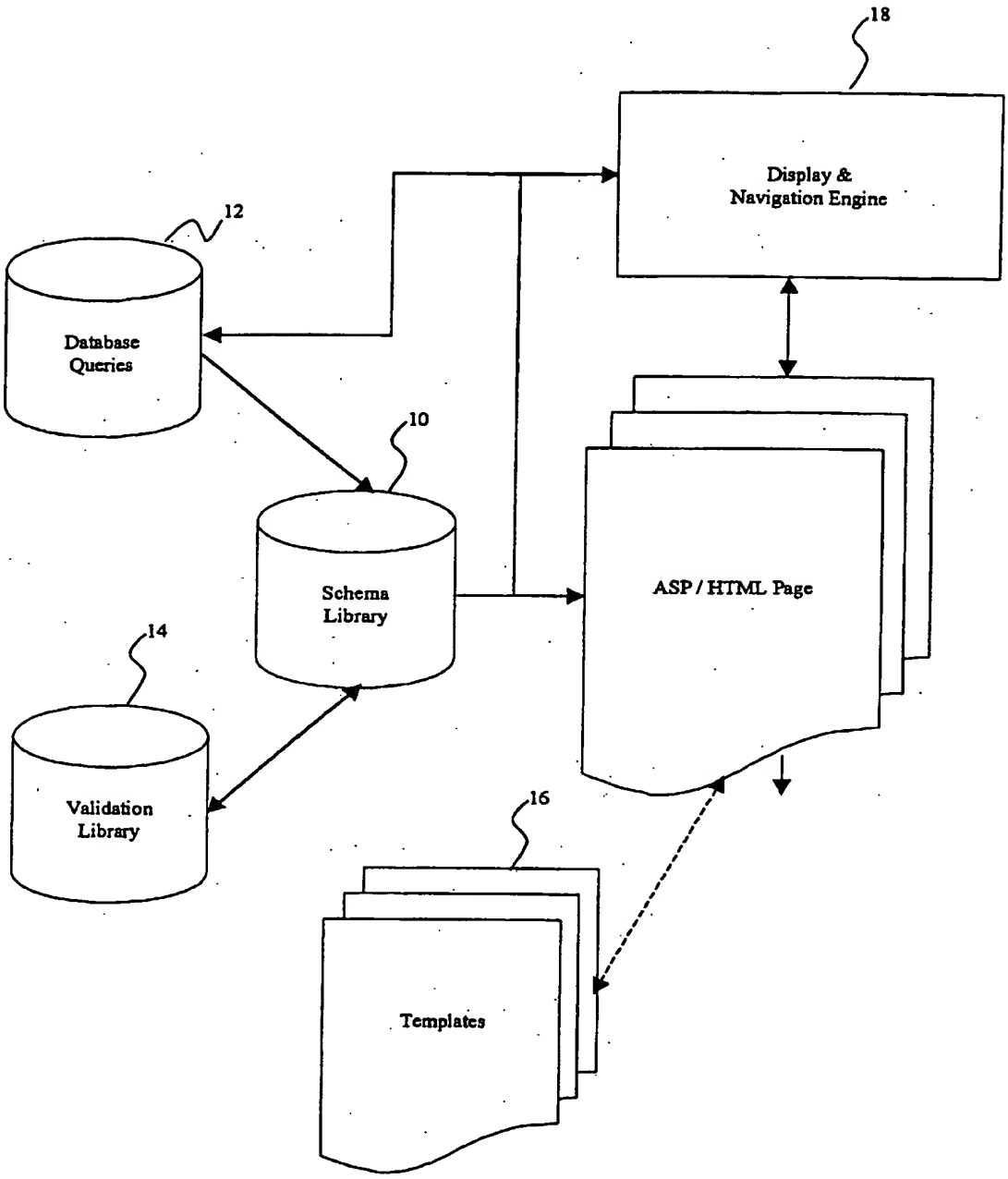

18
Display &
Navigation Engine
12
Database
Queries
10
Schema
Library
ASP / HTML Page
14
Validation
Library
16
Templates

Fig. 1

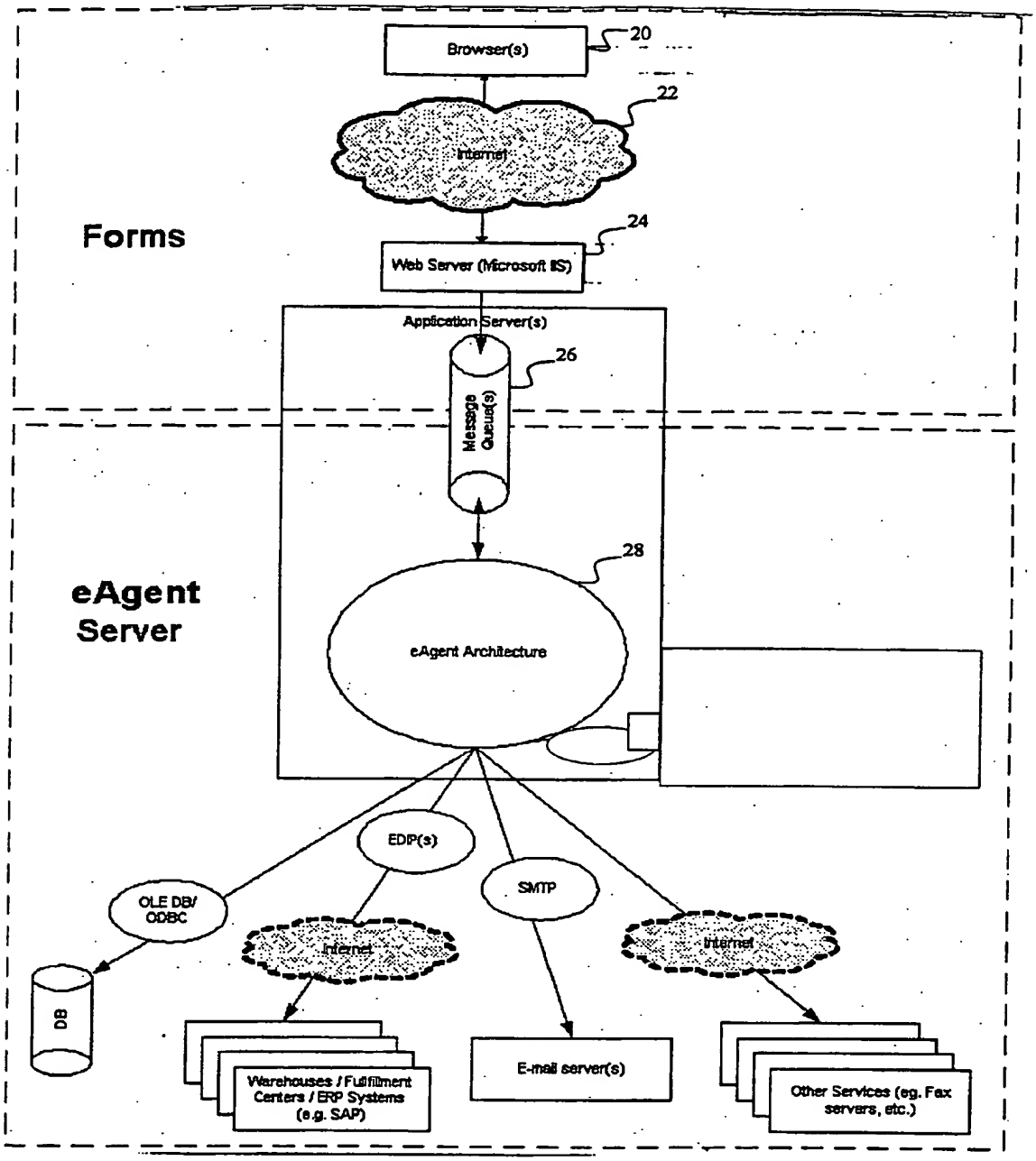

Browser(s)
20
22
Internet
Forms
24
Web Server (Microsoft IIS)
Application Server(s)
26
Message Queue(s)
28
eAgent Server
eAgent Architecture
EDIP(s)
SMTP
OLE DB/ ODBC
Internet
Internet
DB
Warehouses / Fulfillment Centers / ERP Systems (e.g. SAP)
E-mail server(s)
Other Services (eg. Fax servers, etc.)

Fig. 2

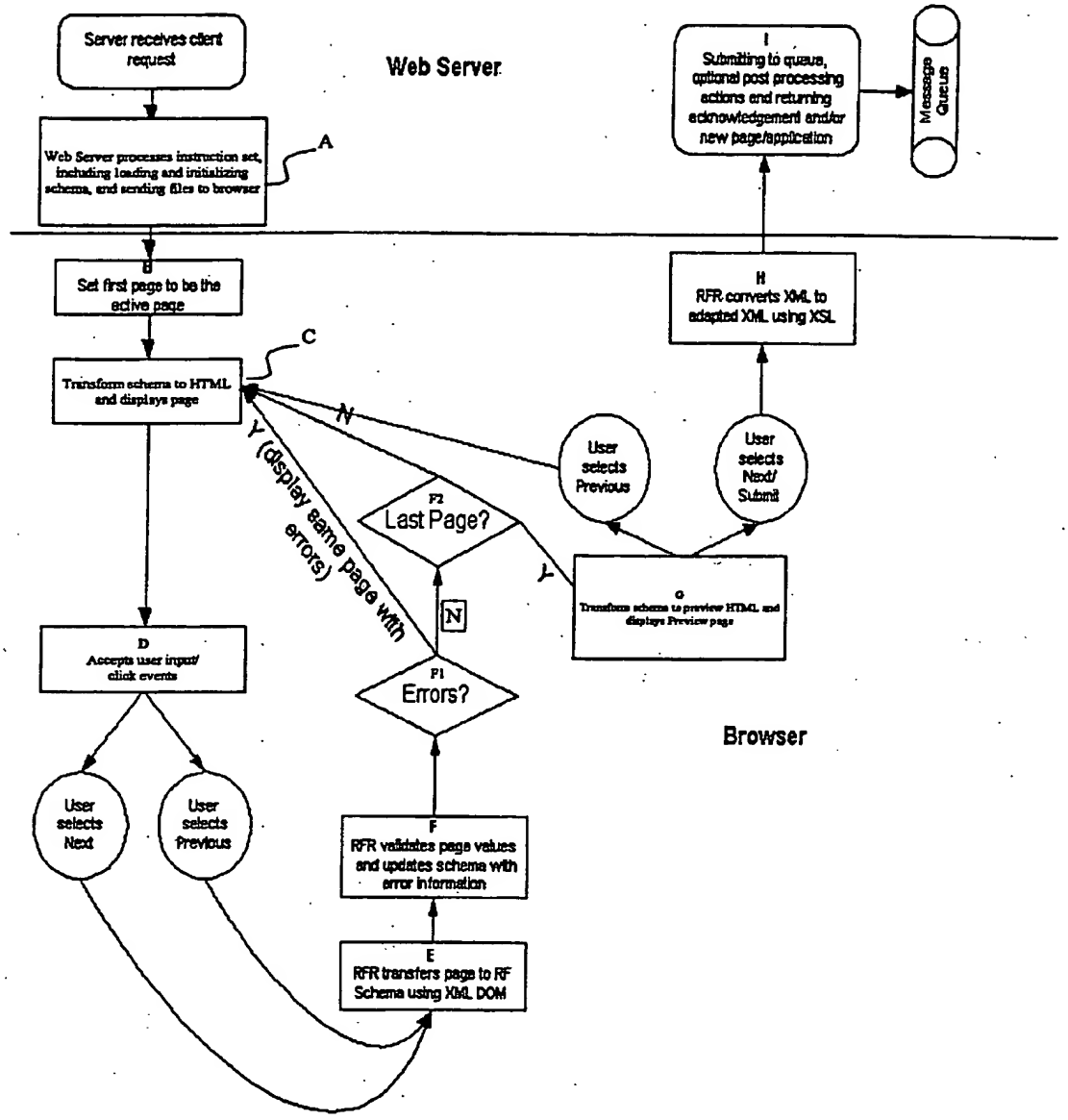

Server receives client request
Web Server
Web Server processes instruction set, including loading and initializing schema, and sending files to browser
A
I
Submitting to queue, optional post processing actions and returning acknowledgement and/or new page/application
Message Queue
B
Set first page to be the active page
H
RFR converts XML to adapted XML using XSL
C
Transform schema to HTML and displays page
N
Y (display same page with errors)
User selects Previous
User selects Next/ Submit
F2
Last Page?
Y
N
O
Transform schema to preview HTML and displays Preview page
D
Accepts user input/ click events
F1
Errors?
Browser
User selects Next
User selects Previous
F
RFR validates page values and updates schema with error information
E
RFR transfers page to RF Schema using XML DOM

Fig. 3

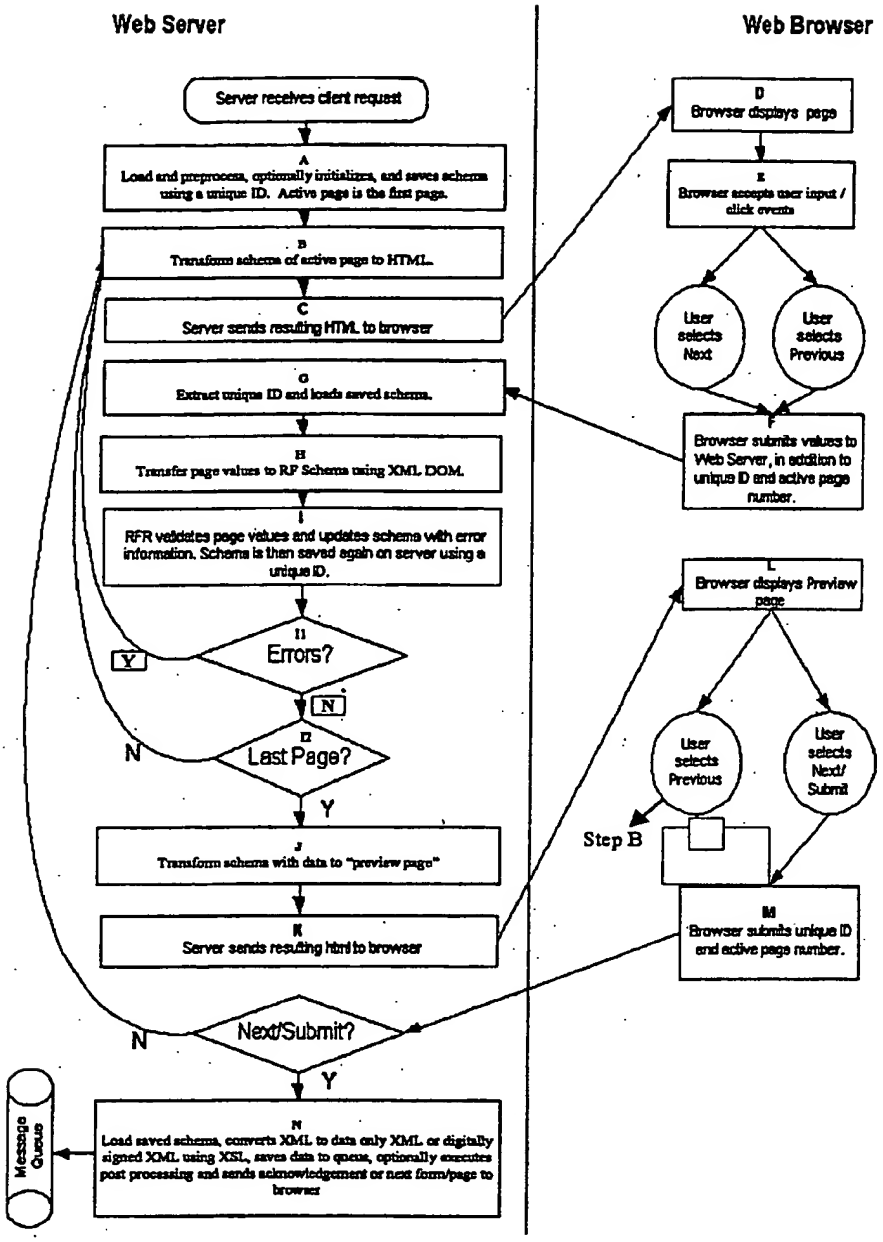

Web Server
Web Browser
Server receives client request
A
Load and preprocess, optionally initialize, and saves schema using a unique ID. Active page is the first page.
B
Transform schema of active page to HTML.
C
Server sends resulting HTML to browser
D
Browser displays page
E
Browser accepts user input / click events
User selects Next
User selects Previous
F
Browser submits values to Web Server, in addition to unique ID and active page number.
G
Extract unique ID and loads saved schema.
H
Transfer page values to RF Schema using XML DOM.
I
RFR validates page values and updates schema with error information. Schema is then saved again on server using a unique ID.
J1
Errors?
Y
N
J2
Last Page?
N
Y
J
Transform schema with data to "preview page"
K
Server sends resulting html to browser
L
Browser displays Preview page
User selects Previous
User selects Next/ Submit
Step B
M
Browser submits unique ID and active page number.
Next/Submit?
N
Y
N
Load saved schema, converts XML to data only XML or digitally signed XML using XSL, saves data to queue, optionally executes post processing and sends acknowledgement or next form/page to browser
Message Queue

Fig. 4

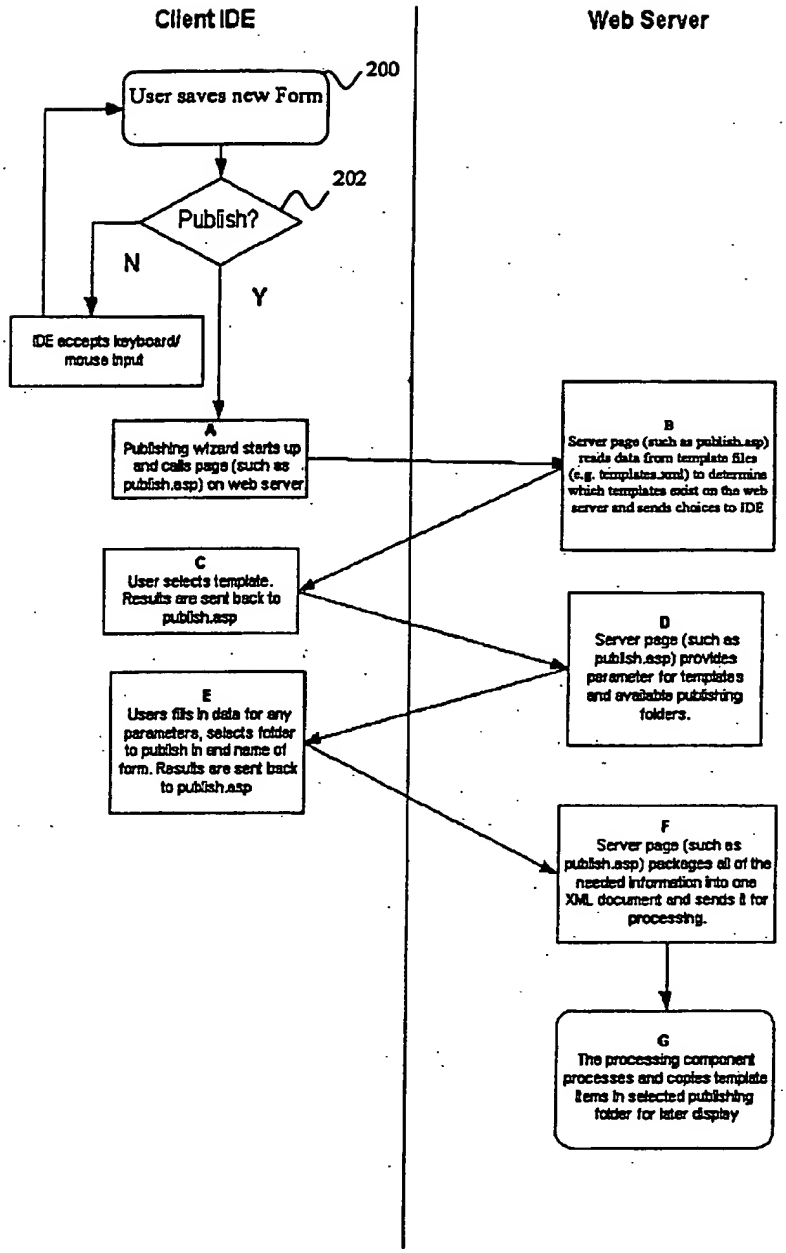

Client IDE
Web Server
User saves new Form
200
Publish?
202
N
Y
IDE accepts keyboard/ mouse input
A
Publishing wizard starts up and calls page (such as publish.asp) on web server
B
Server page (such as publish.asp) reads data from template files (e.g. templates.xml) to determine which templates exist on the web server and sends choices to IDE
C
User selects template. Results are sent back to publish.asp
D
Server page (such as publish.asp) provides parameter for templates and available publishing folders.
E
Users fills in data for any parameters, selects folder to publish in and name of form. Results are sent back to publish.asp
F
Server page (such as publish.asp) packages all of the needed information into one XML document and sends it for processing.
G
The processing component processes and copies template items in selected publishing folder for later display

Fig. 5

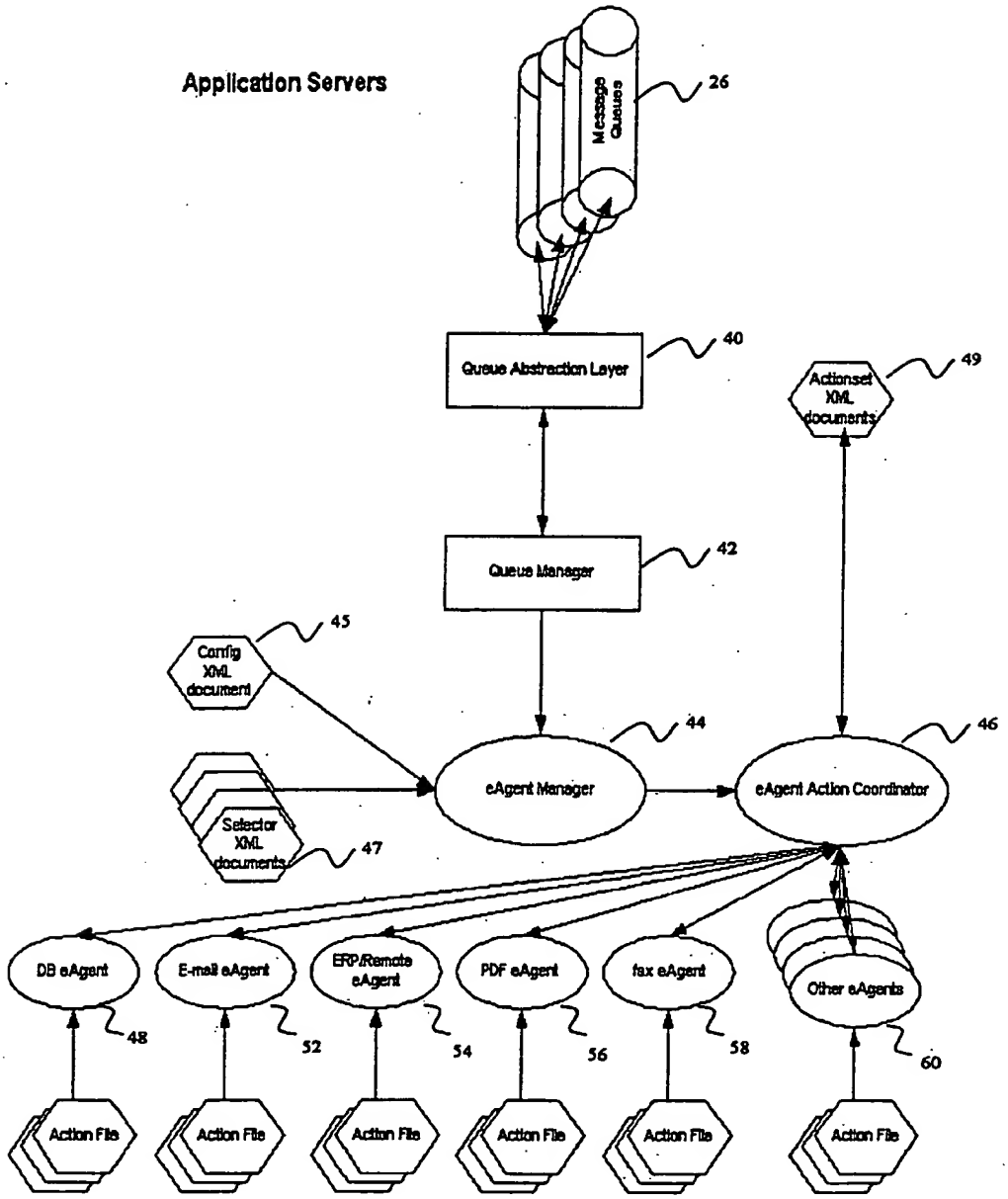

Application Servers
Message Queues
26
Queue Abstraction Layer
40
Actionset XML documents
49
Queue Manager
42
Config XML document
45
eAgent Manager
44
eAgent Action Coordinator
46
Selector XML documents
47
DB eAgent
48
E-mail eAgent
52
ERP/Remote eAgent
54
PDF eAgent
56
fax eAgent
58
Other eAgents
60
Action File
Action File
Action File
Action File
Action File
Action File

Fig. 6

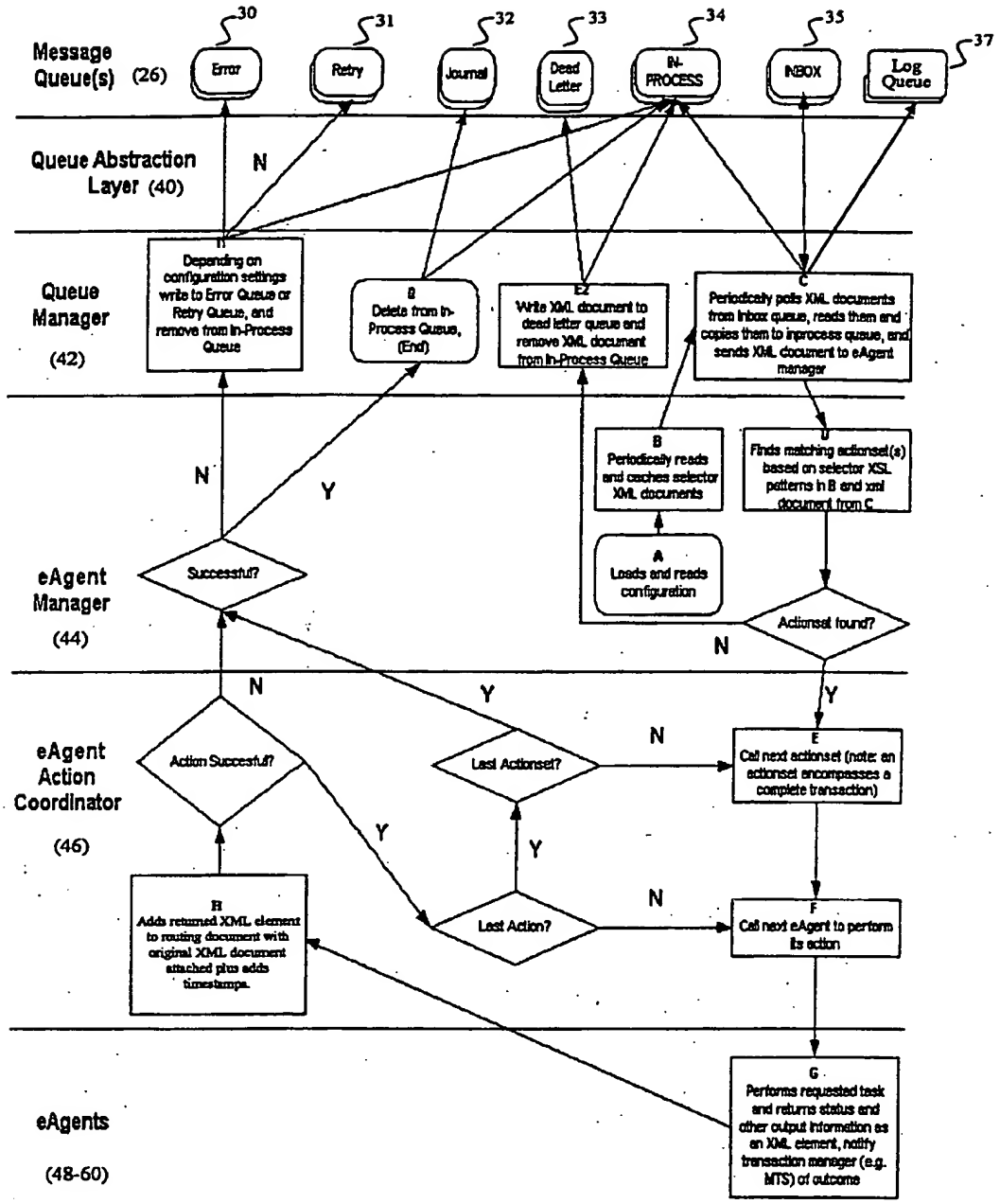

Message Queue(s) (26)
30
Error
31
Retry
32
Journal
33
Dead Letter
34
IN-PROCESS
35
INBOX
Log Queue
37
Queue Abstraction Layer (40)
N
Queue Manager (42)
H
Depending on configuration settings write to Error Queue or Retry Queue, and remove from In-Process Queue
B
Delete from In-Process Queue, (End)
E2
Write XML document to dead letter queue and remove XML document from In-Process Queue
C
Periodically polls XML documents from Inbox queue, reads them and copies them to inprocess queue, and sends XML document to eAgent manager
B
Periodically reads and caches selector XML documents
D
Finds matching actionset(s) based on selector XSL patterns in B and xml document from C
A
Loads and reads configuration
N
Y
eAgent Manager (44)
Successful?
Actionset found?
N
Y
eAgent Action Coordinator (46)
Action Successful?
Last Actionset?
Last Action?
E
Call next actionset (note: an actionset encompasses a complete transaction)
F
Call next eAgent to perform its action
H
Adds returned XML element to routing document with original XML document attached plus adds timestamps.
eAgents (48-60)
G
Performs requested task and returns status and other output information as an XML element, notify transaction manager (e.g. MTS) of outcome

Fig. 7

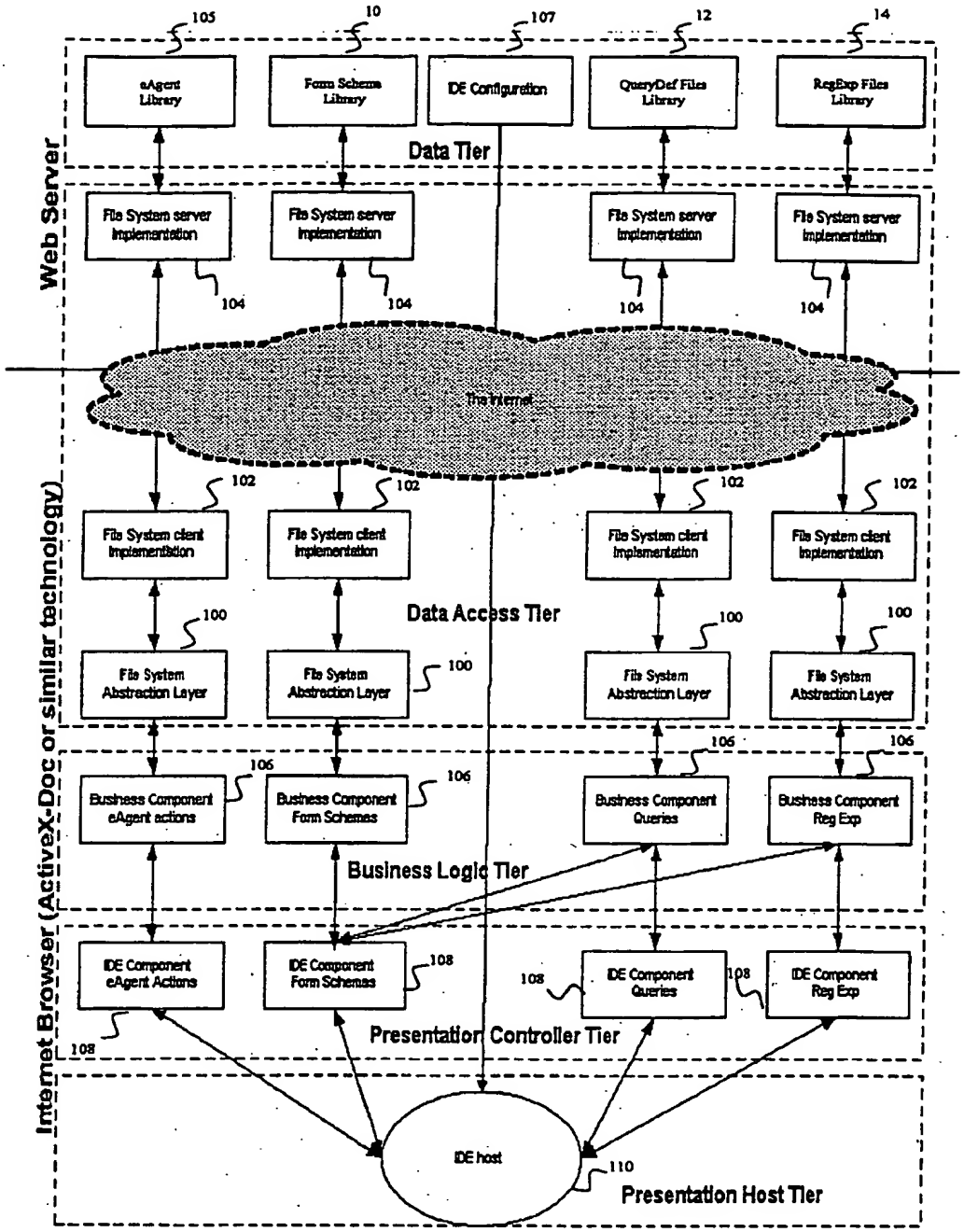

105
10
107
12
14
eAgent Library
Form Schema Library
IDE Configuration
QueryDef Files Library
RegExp Files Library
Data Tier
Web Server
File System server Implementation
104
The Internet
102
File System client Implementation
100
Data Access Tier
File System Abstraction Layer
106
Business Component eAgent actions
Business Component Form Schemas
Business Component Queries
Business Component Reg Exp
Business Logic Tier
IDE Component eAgent Actions
IDE Component Form Schemas
108
IDE Component Queries
IDE Component Reg Exp
Presentation Controller Tier
Internet Browser (ActiveX-Doc or similar technology)
IDE host
110
Presentation Host Tier

Fig. 8

Forms - Schema

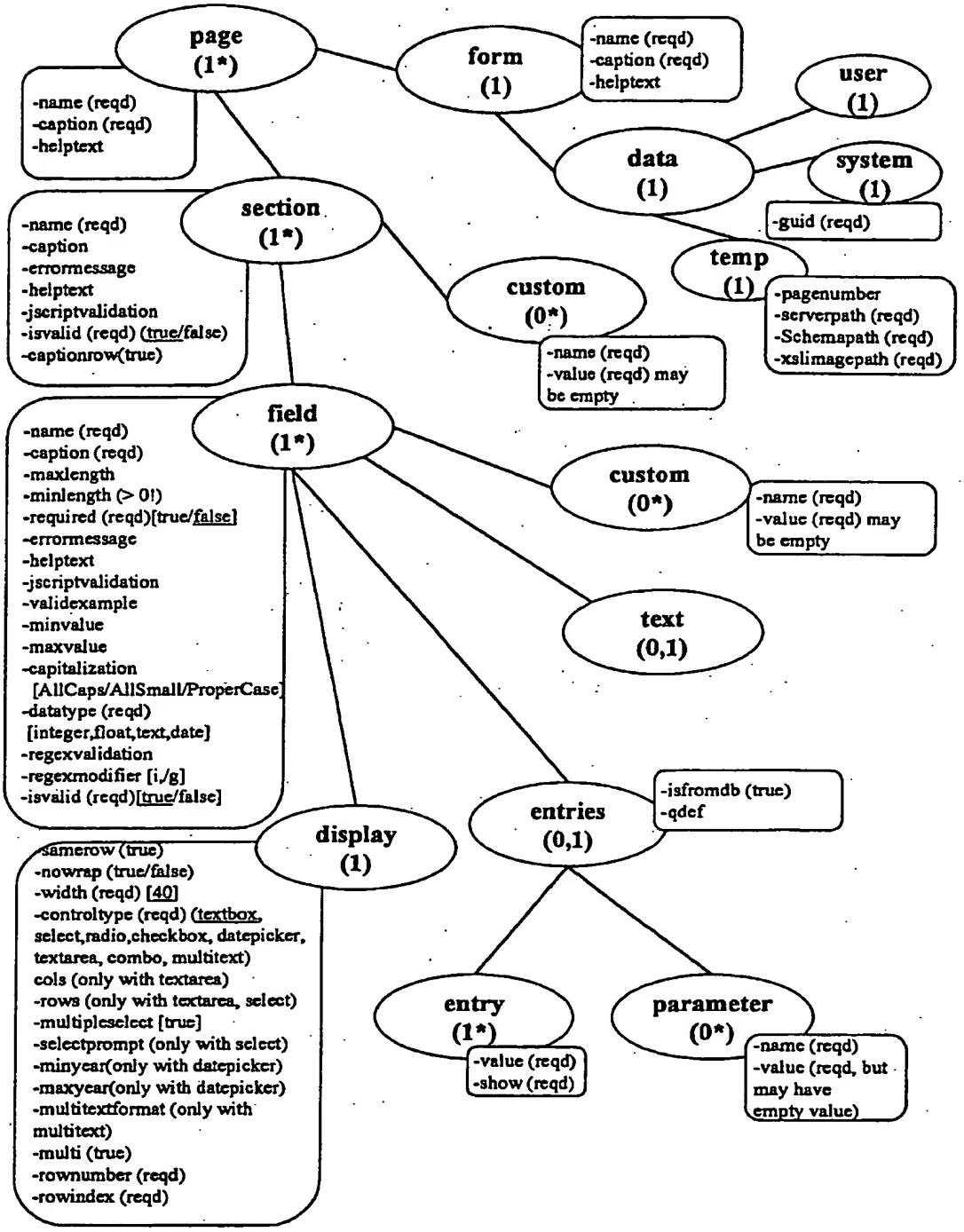


Fig. 9

Regular Expressions - Schema

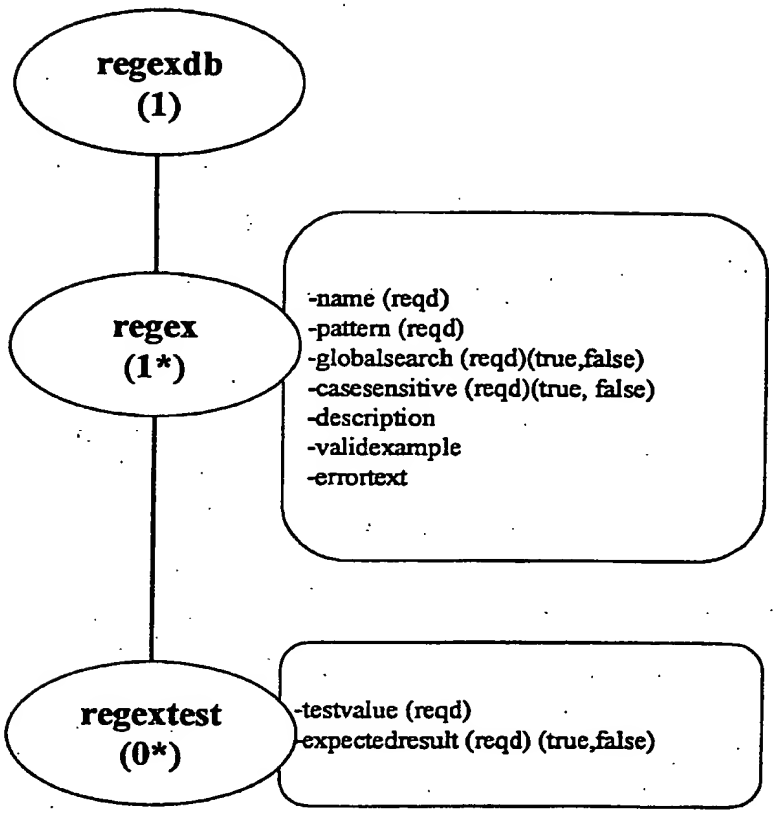


Fig. 10

Select Query - Schema

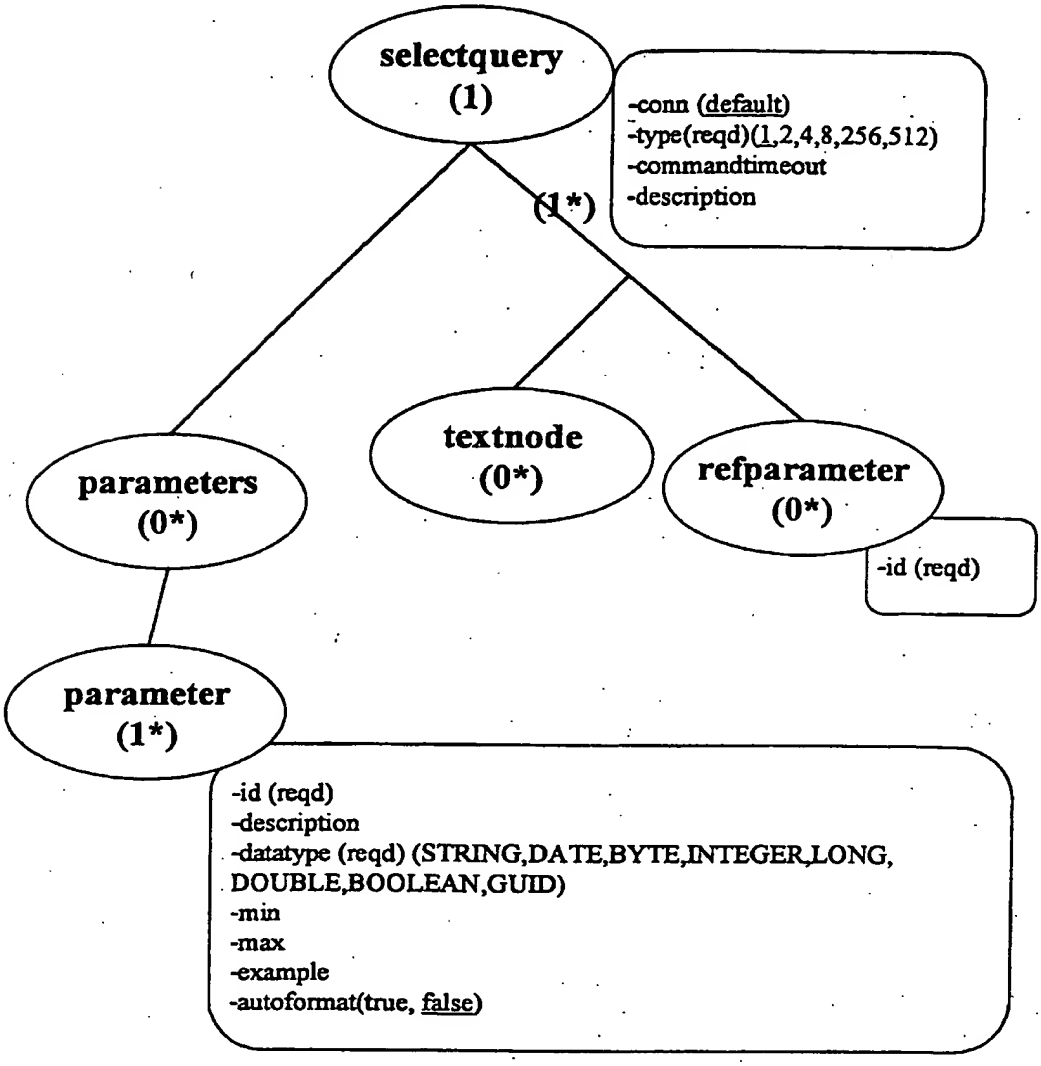


Fig. 11

Templates - Schema

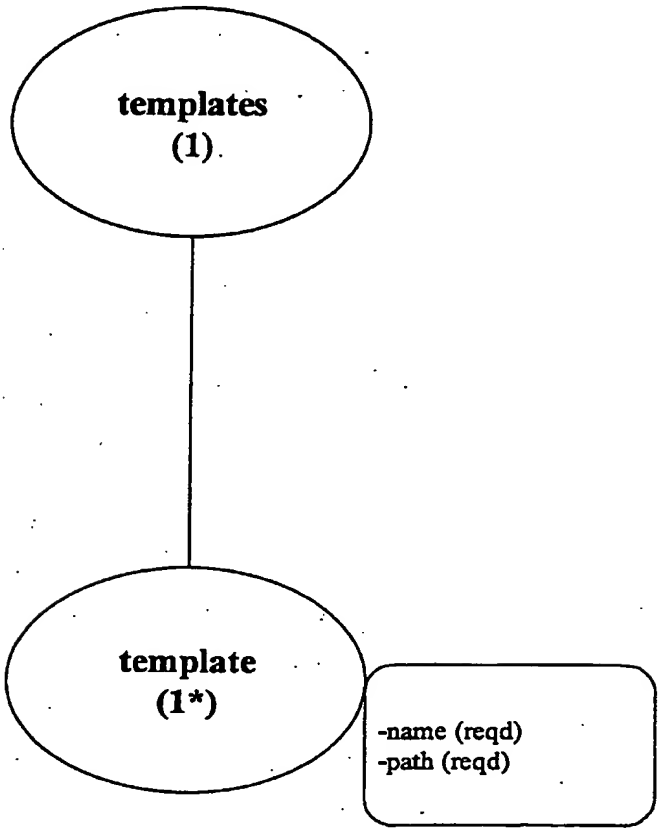


Fig. 12

Template - Schema

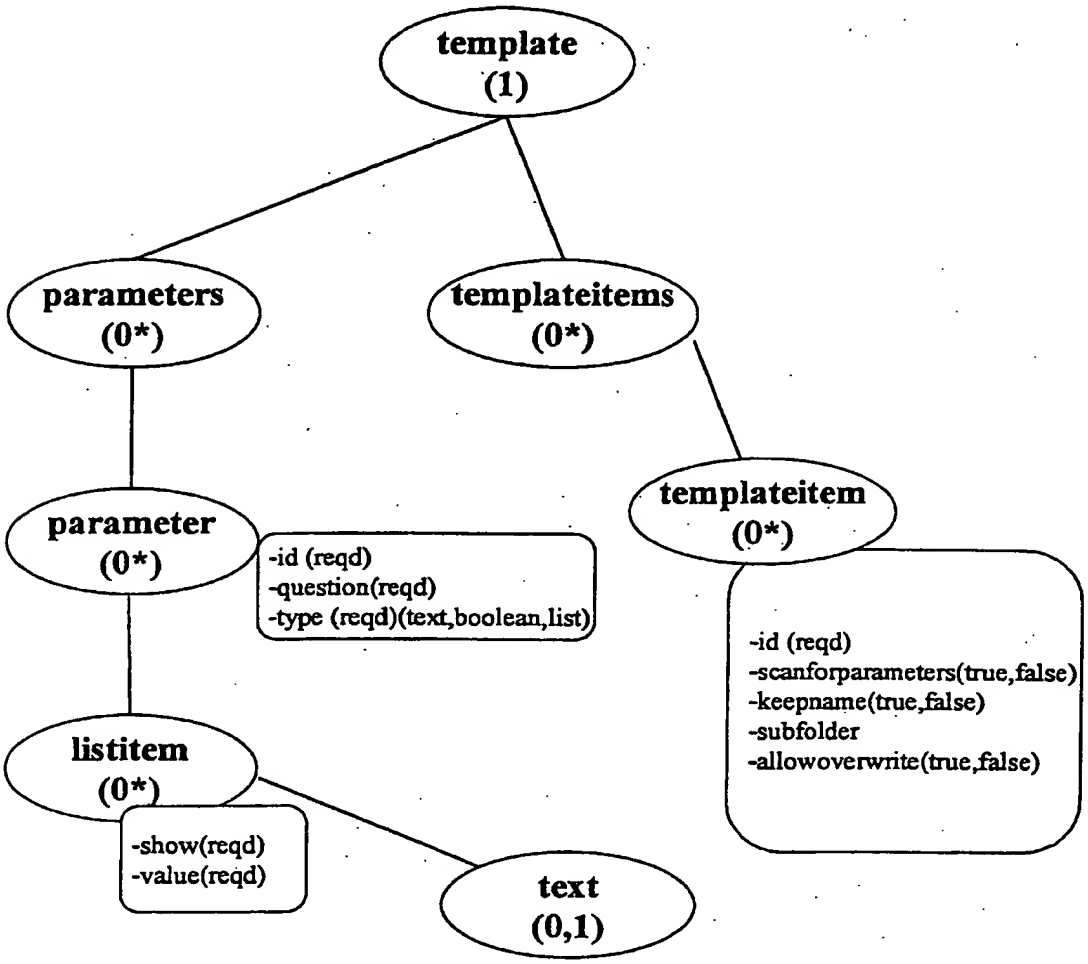


Fig. 13

Publishfolders - Schema

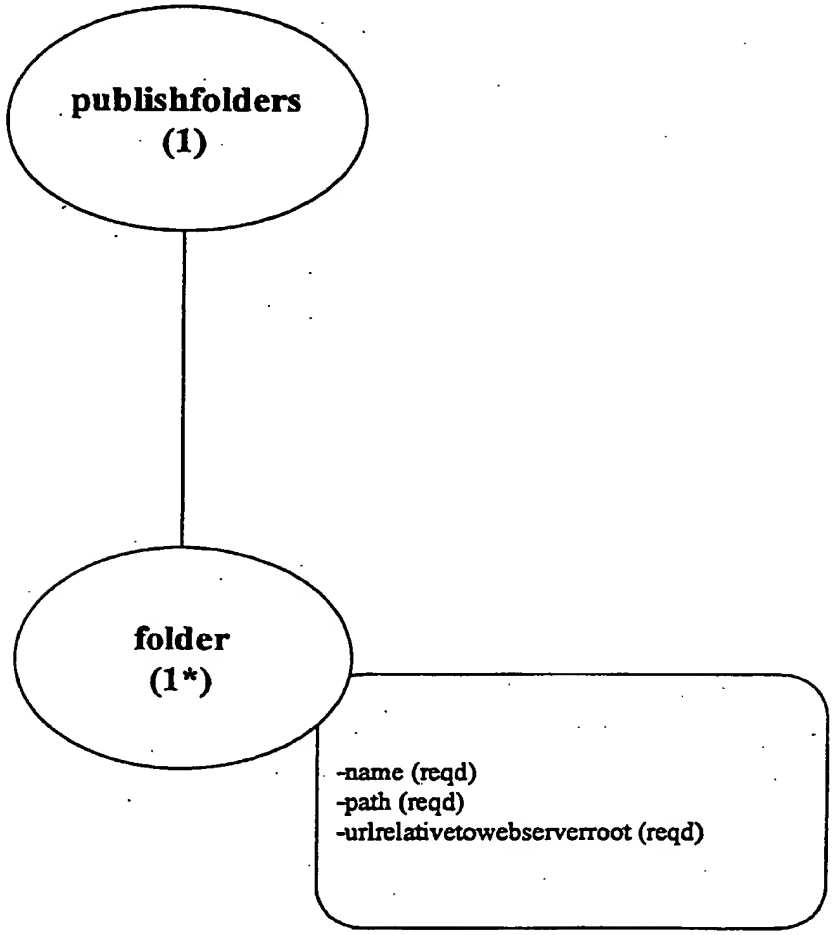


Fig. 14

IDE configuration - Schema

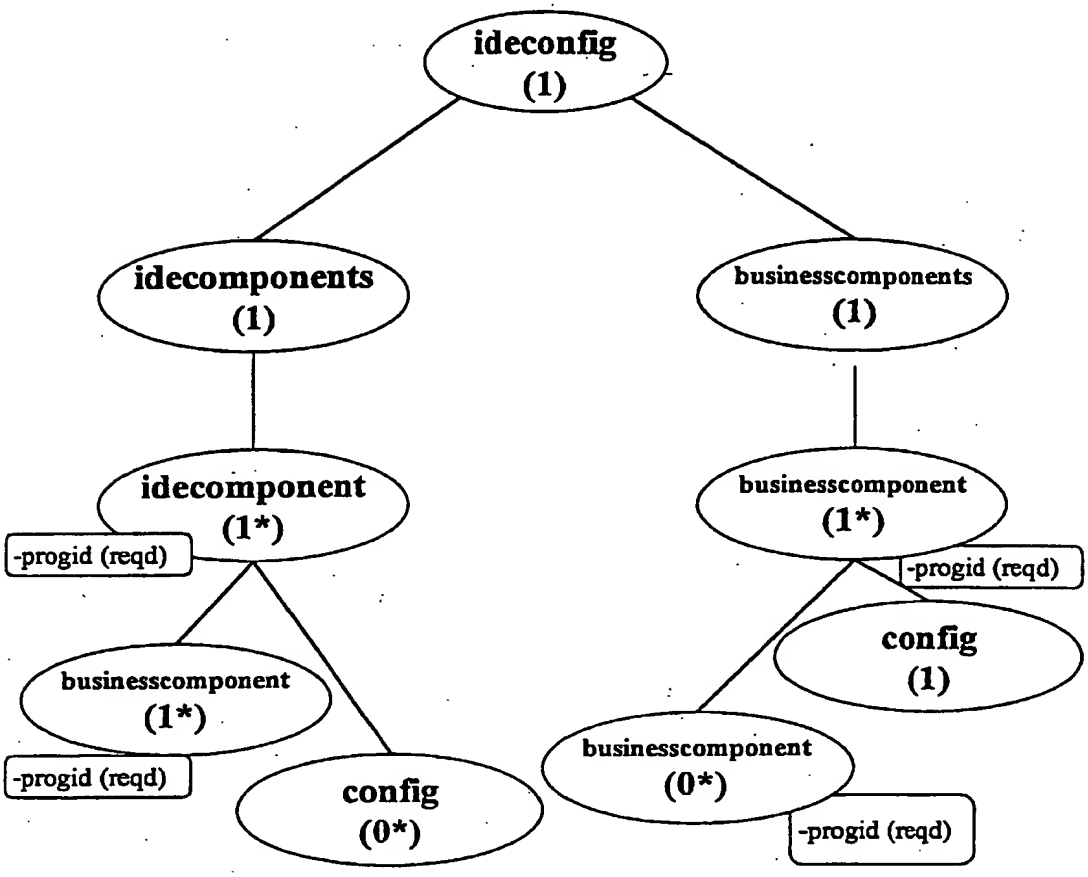


Fig. 15

XML Selector - Schema

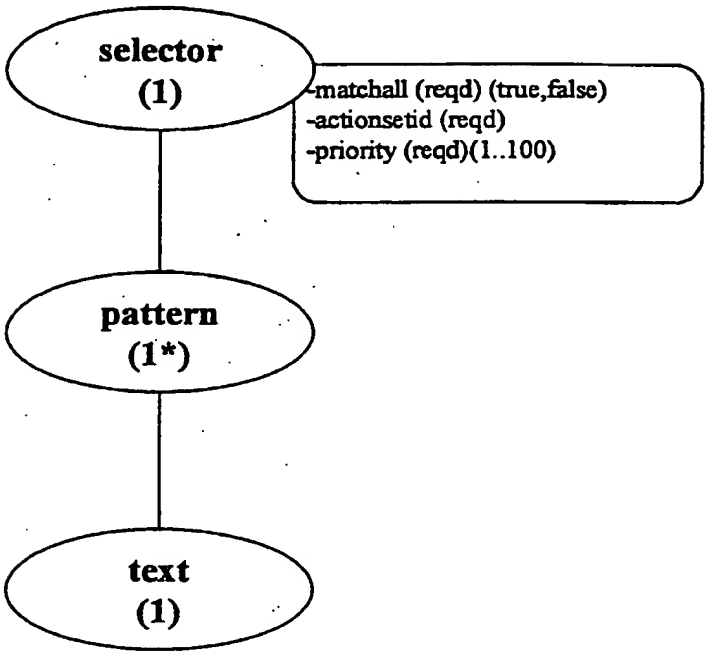


Fig. 16

ActionSet - Schema

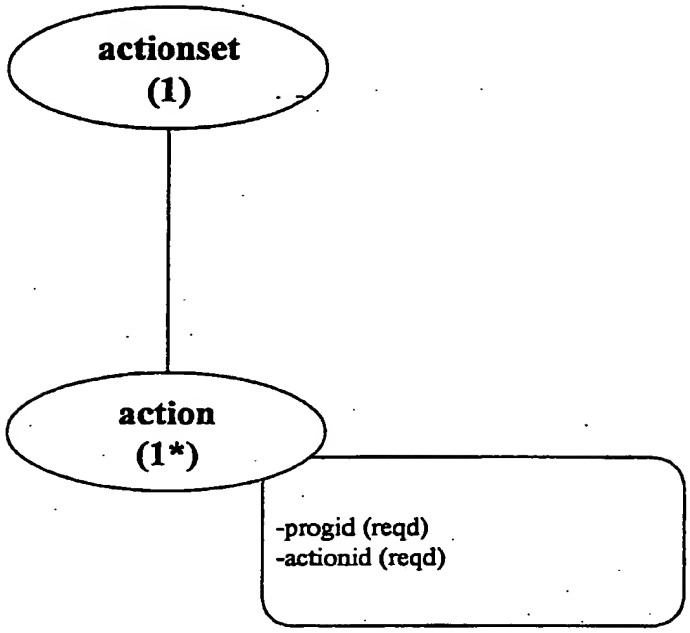


Fig. 17

Action Query eAgent - Schema

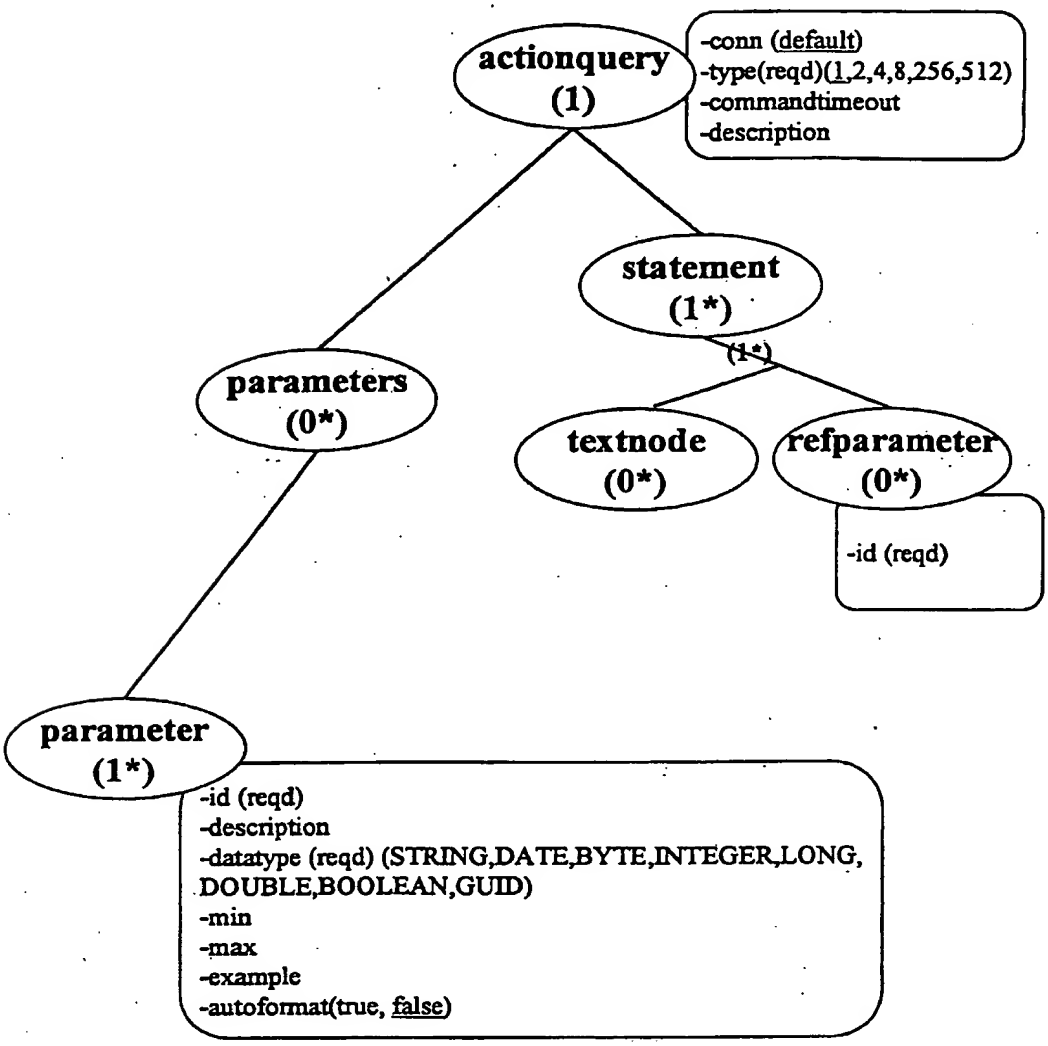


Fig. 18

Queue Config - Schema

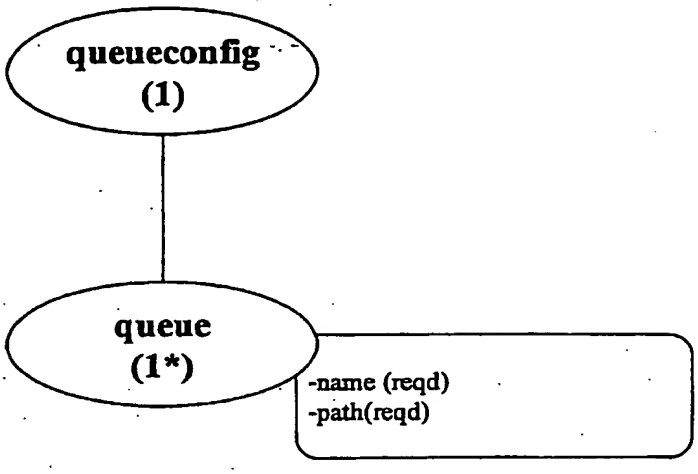


Fig. 19

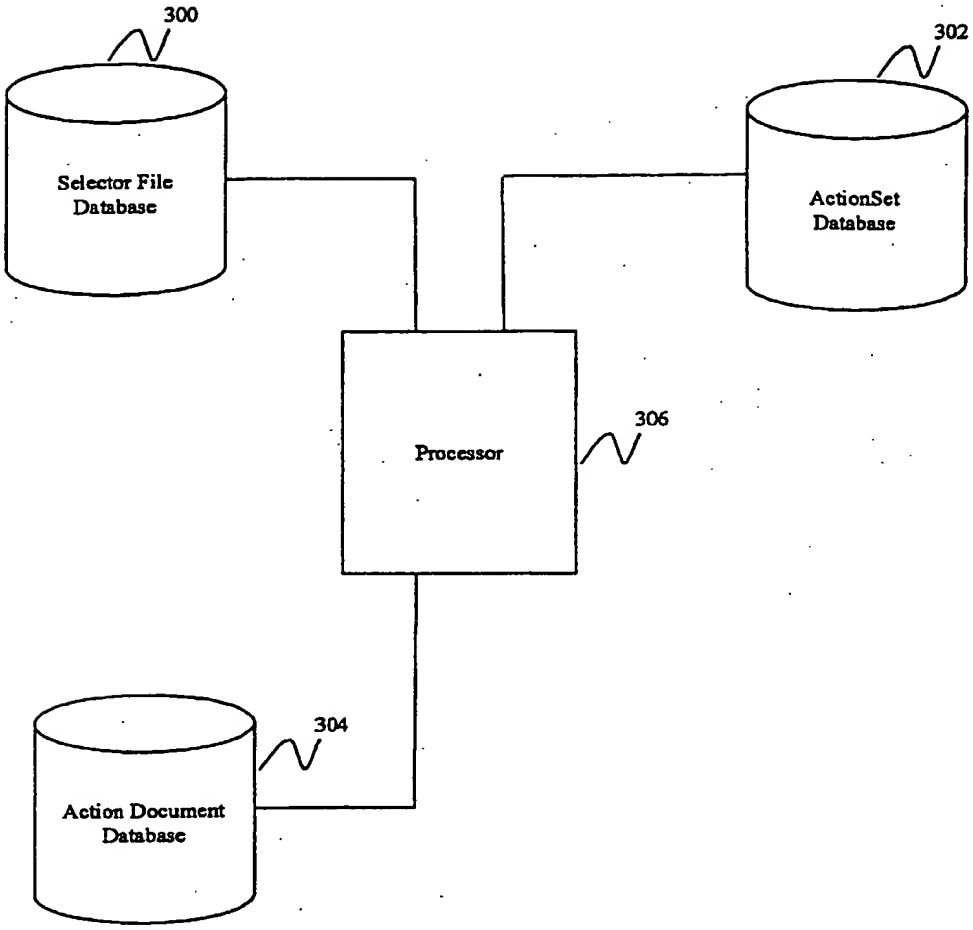

300
Selector File Database
302
ActionSet Database
Processor
306
304
Action Document Database

Fig. 20

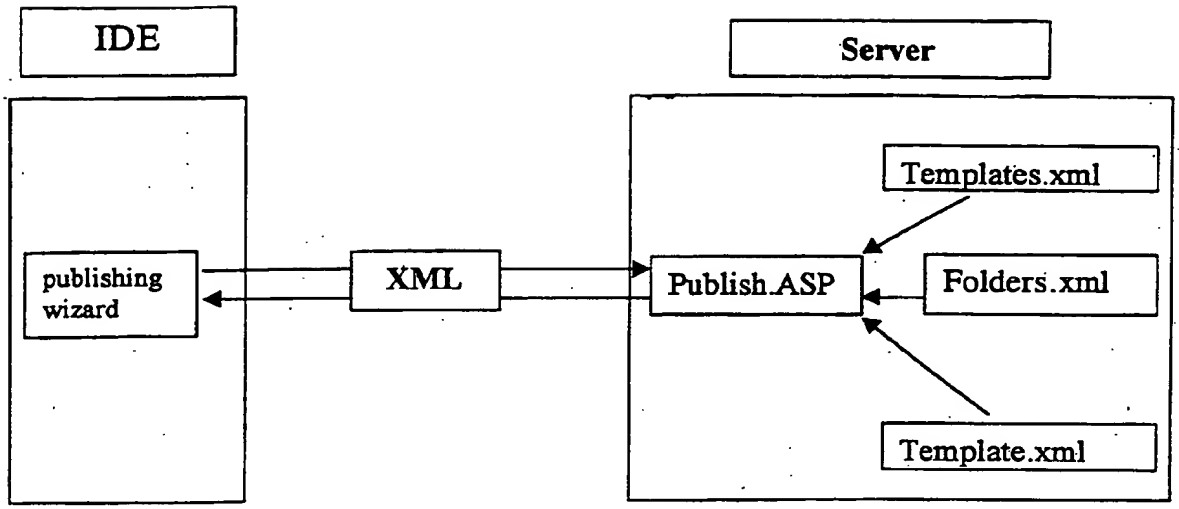

IDE
Server
Templates.xml
publishing wizard
XML
Publish.ASP
Folders.xml
Template.xml

Fig. 21